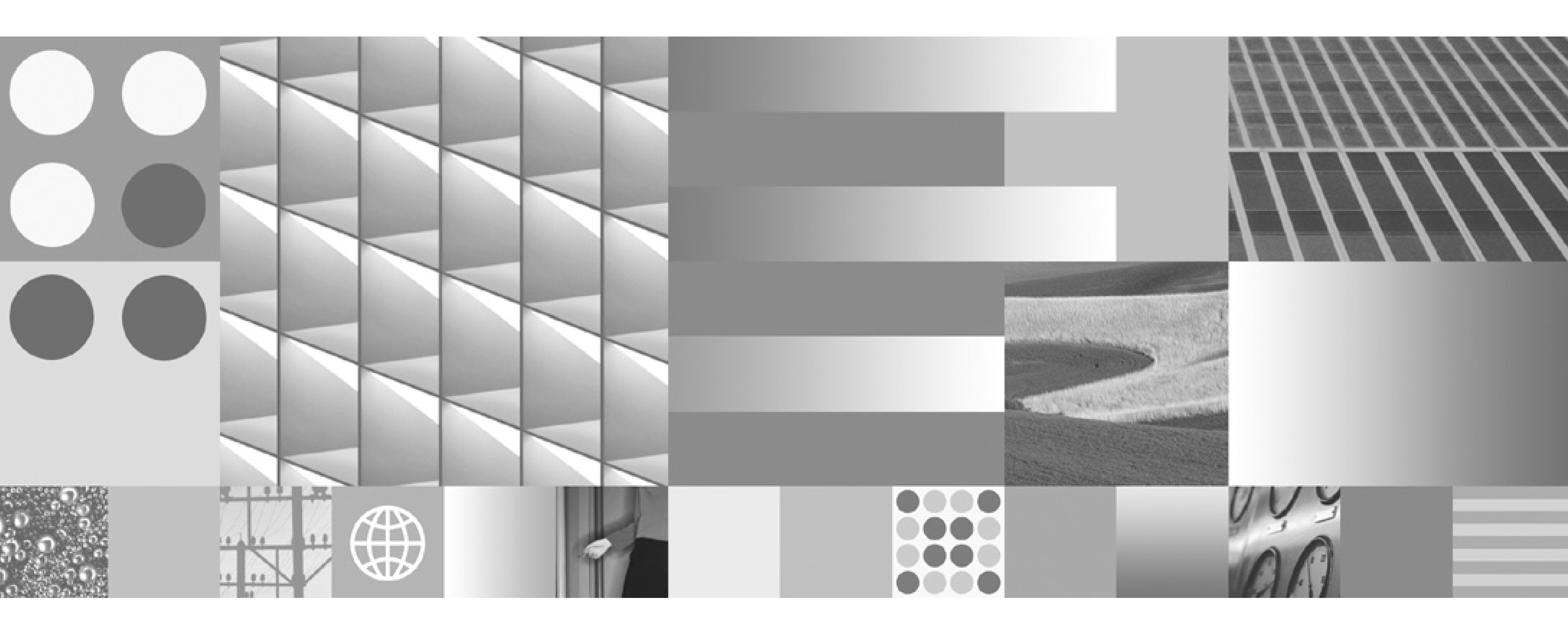
IBM DB2 9.7
Linux 版、UNIX 版和 Windows 版



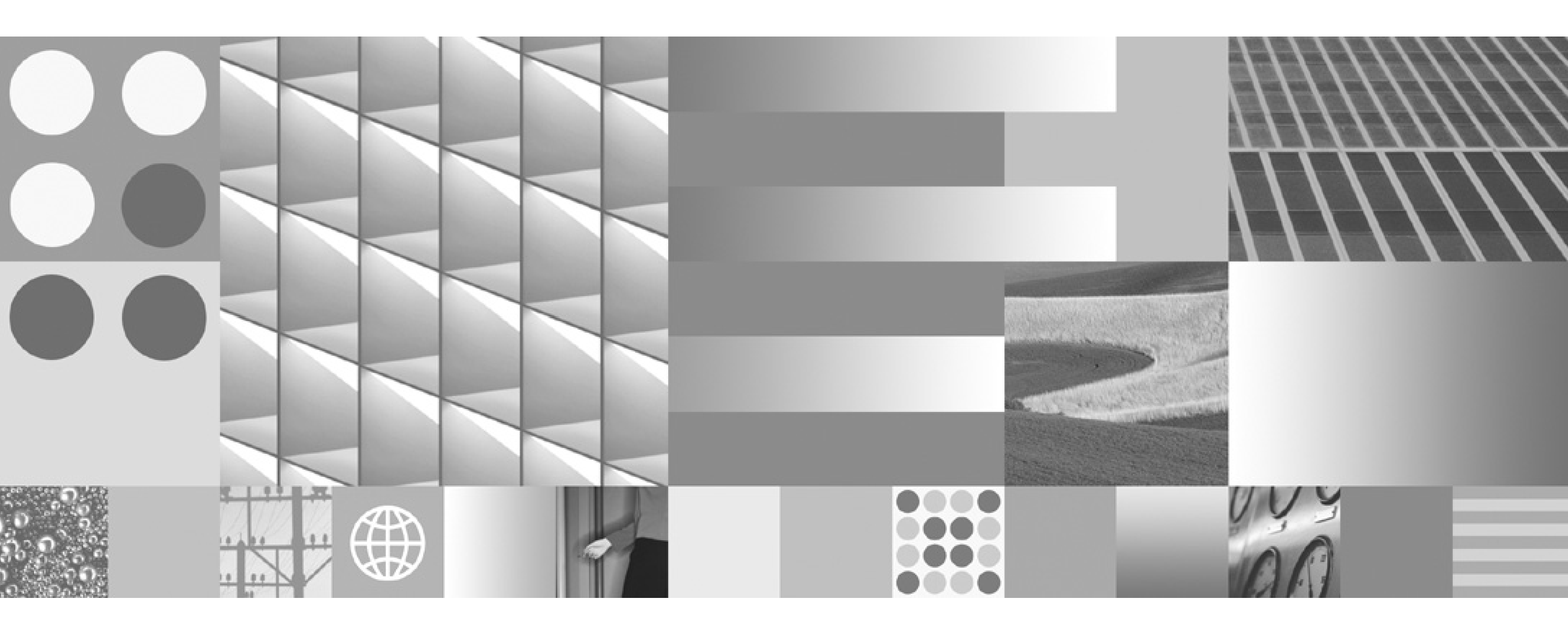


pureXML 指南

S151-1180-00

IBM DB2 9.7
Linux 版、UNIX 版和 Windows 版





pureXML 指南

S151-1180-00

注意

使用此信息及其支持的产品前，请先阅读第 447 页的附录 E，『声明』下的常规信息。

修订版声明

此文档包含 IBM 的所有权信息。它在许可协议中提供，且受版权法的保护。本出版物中包含的信息不包括对任何产品的保证，且提供的任何语句都不需要如此解释。

您可在线或通过当地的 IBM 代表处订购 IBM 出版物。

- 要在线订购出版物，请转至 IBM 出版物中心，网址为：www.ibm.com/shop/publications/order
- 要查找当地的 IBM 代表处，请转至 IBM 全球联系人目录，网址为：www.ibm.com/planetwide

要从美国或加拿大的 DB2 市场和销售部订购 DB2 出版物，请致电 1-800-IBM-4YOU（426-4968）。

当您向 IBM 发送信息时，即同意授予 IBM 独一无二的权力以它认为适当且不会对您造成任何影响的方式使用或分发该信息。

目录

第 7 章 更新 XML 数据 165

第 8 章 XML 模式存储库. 171

第 9 章 XML 数据移动 187

第 10 章 应用程序编程语言支持 203

第 11 章 XML 性能 261

第 12 章 XML 数据编码 273

关于本书

pureXML™ 指南描述如何在 DB2® 数据库中使用 XML 数据。它将告诉您有关 XML 数据类型和 XML 存储器的知识，如何使用 SQL 和 XQuery 语言来使用 XML 数据，以及如何为性能建立索引 XML 数据。其他主题包括 pureXML 应用程序发展，数据移动和把 XML 数据分解为相关格式。

第 1 章 pureXML 概述 – DB2 作为 XML 数据库

pureXML 功能部件允许您将格式良好的 XML 文档存储在具有 XML 数据类型的数据库表列中。通过将 XML 数据存储在 XML 列中，数据可保持其本机分层结构形式，而不是将其作为文本存储或映射为其他数据模型。

因为 pureXML 数据存储已完全集成，所以可利用现有 DB2 数据库服务器功能来访问和管理存储的 XML 数据。

将 XML 数据以其本机分层结构形式来存储可以使 XML 的搜索、检索和更新效率更高。XQuery、SQL 或上述二者的组合可用于查询和更新 XML 数据。返回 XML 数据或采用 XML 自变量的 SQL 函数（称为 SQL/XML 函数）还能根据从数据库中检索的值来构造或发布 XML 数据。

查询和更新

可使用下列方法查询和更新存储在 XML 列中的 XML 文档:

XQuery

XQuery 是用于解释、检索和修改查询 XML 数据的通用语言。DB2 数据库服务器允许直接调用 XQuery 或从 SQL 调用 XQuery。因为 XML 数据存储在 DB2 表和视图中，所以提供了一些函数，用于通过直接命名表或视图或通过指定 SQL 查询从指定的表和视图中抽取 XML 数据。XQuery 支持各种用于处理 XML 数据、更新元素和属性之类的 XML 对象和构造新 XML 对象的表达式。XQuery 的编程接口提供了类似于 SQL 的功能，用于执行查询并检索结果。

SQL 语句和 SQL/XML 函数

许多 SQL 语句支持 XML 数据类型。这使得您能够对 XML 数据执行许多常见数据库操作，例如，创建具有 XML 列的表、将 XML 列添加至现有表、创建基于 XML 列的索引、对具有 XML 列的表创建触发器以及插入、更新或删除 XML 文档。DB2 数据库服务器支持的一组 SQL/XML 函数、表达式和规范利用 XML 数据类型。

可以从 SQL 查询中调用 XQuery。在这种情况下，SQL 查询可以将数据以绑定变量的形式传递至 XQuery。

应用程序开发

许多编程语言通过 SQL 和外部过程提供了对应用程序开发的支持:

编程语言支持

对新的 pureXML 功能的应用程序开发支持使得应用程序能够组合 XML 和关系数据的访问和存储。下列编程语言支持 XML 数据类型:

- C 或 C++（嵌入式 SQL 或 DB2 CLI）
- COBOL
- Java™（JDBC 或 SQLJ）
- C# 和 Visual Basic（IBM® 数据服务器 .NET 提供程序）
- PHP

- Perl

SQL 和外部过程
: 通过在 CREATE PROCEDURE 参数特征符中包含 XML 数据类型参数，可将 XML 数据传递至 SQL 过程和外部过程。现有的过程功能支持围绕 SQL 语句实现过程逻辑流，它们产生或利用 XML 值以及变量中的 XML 数据值临时存储器。

管理

pureXML 功能提供用于管理 XML 文档的 URI 依赖关系的存储库并允许用于数据库管理的 XML 数据移动:

XML 模式存储库（XSR）
: XML 模式存储库（XSR）是用于存放在处理 XML 列中存储的 XML 实例文档时所需的所有 XML 工件的存储库。它存储 XML 文档中引用的 XML 模式、DTD 和外部实体。

导入、导出和装入实用程序
: 导入、导出和装入实用程序已更新为支持本机 XML 数据类型。这些实用程序象处理 LOB 数据一样来处理 XML 数据: 这两种类型的数据都存储在实际的表之外。已更新的 db2Import、db2Export 和 db2Load API 还提供了用于导入、导出和装入 XML 数据的应用程序开发支持。这些已更新的实用程序允许移动存储在 XML 列中的 XML 文档数据，这类似于对关系数据的数据移动支持。

性能

使用存储在 XML 列中的 XML 文档时，可使用提高性能的功能部件:

基于 XML 数据的索引
: 对存储在 XML 列中的数据提供了索引支持。使用基于 XML 数据的索引可提高针对 XML 文档发出的查询的效率。与关系索引类似，XML 数据索引对列建立索引。 但是，它们的区别在于关系索引是对整个列建立索引，而XML 数据索引只是对部分列建立索引。通过指定 XML 模式（它是受限的 XPath 表达式），可指明要对 XML 列的哪些部分建立索引。

优化器
: 更新了优化器，以支持针对 XML 数据和关系数据对 SQL 函数、XQuery 函数和嵌入了 XQuery 的 SQL/XML 函数进行求值。优化器采用通过 XML 数据以及来自基于 XML 数据的索引的数据所收集的统计信息生成有效的查询执行计划。

说明和 Visual Explain
: 已更新说明设施和 Visual Explain GUI 工具，以便支持用于查询 XML 数据的 SQL 增强功能并支持 XQuery 表达式。对说明设施和 Visual Explain GUI 工具的这些更新允许您快速查看 DB2 数据库服务器针对 XML 数据对查询语句进行求值的方式。

工具

一些工具支持 XML 数据类型，这些工具包括控制中心、命令行处理器、IBM Data Studio 和 DB2 Development Add-In for Microsoft® Visual Studio .NET。

带注释的 XML 模式分解

pureXML 使您能够将采用分层格式的 XML 数据作为 XML 存储和访问，但有时可能需要将 XML 数据作为关系数据来访问。带注释的 XML 模式分解根据 XML 模式中指定的注释来分解文档。

XML 数据类型

此数据类型用于定义表中存储 XML 值的列，这些列中存储的所有 XML 值必须是格式良好的 XML 文档。引入此本机 XML 数据类型能够将格式良好的 XML 文档存储在数据库中其他关系数据旁边的本机分层格式中。

使用内部表示来处理 XML 值，内部表示不是字符串，并且不能直接与字符串值进行比较。通过使用 XMLSERIALIZE 函数或将 XML 值绑定至类型为 XML、字符串或二进制的应用程序变量，可以将 XML 值变换成表示 XML 文档的已序列化字符串值。同样，通过使用 XMLPARSE 函数或将应用程序字符串、二进制或 XML 应用程序类型绑定至 XML 值，可以将表示 XML 文档的字符串值变换为 XML 值。在涉及 XML 列的 SQL 数据更改语句（如 INSERT）中，通过使用已插入的 XMLPARSE 函数将表示 XML 文档的字符串或二进制值变换成 XML 值。与应用程序字符串和二进制数据类型交换时，可以隐式解析或序列化 XML 值。

在结构上对数据库中的 XML 值的大小没有限制。但是，请注意，与 DB2 数据库服务器交换的已序列化 XML 数据的大小限制为 2 GB。

可使用 SQL 数据操作语句插入、更新和删除 XML 文档。XML 模式存储库（XSR）支持通常在插入或更新期间针对 XML 模式验证 XML 文档。DB2 数据库系统还提供了用于构造和查询 XML 值以及导出和导入 XML 数据的机制。可以对 XML 列定义 XML 数据索引，从而改善 XML 数据的搜索性能。可以通过各种应用程序接口将表或视图列中的 XML 数据作为已序列化的字符串数据检索。

XML 输入和输出概述

DB2 数据库服务器用于管理关系数据和 XML 数据，它提供了各种方法来输入和输出 XML 文档。

XML 文档存储在定义为 XML 数据类型的列中。一个 XML 列中的每一行都存储单个格式良好的 XML 文档。存储的文档分层保存，并且保留了 XML 数据模型；文档未存储为文本或映射至另一数据模型。

可以在包含其他类型的列（这些列保存关系数据）的表中定义 XML 列；可以为单个表定义多个 XML 列。

输入

第 4 页的图 1 显示了可用于将 XML 数据放入数据库系统中的各种方法。

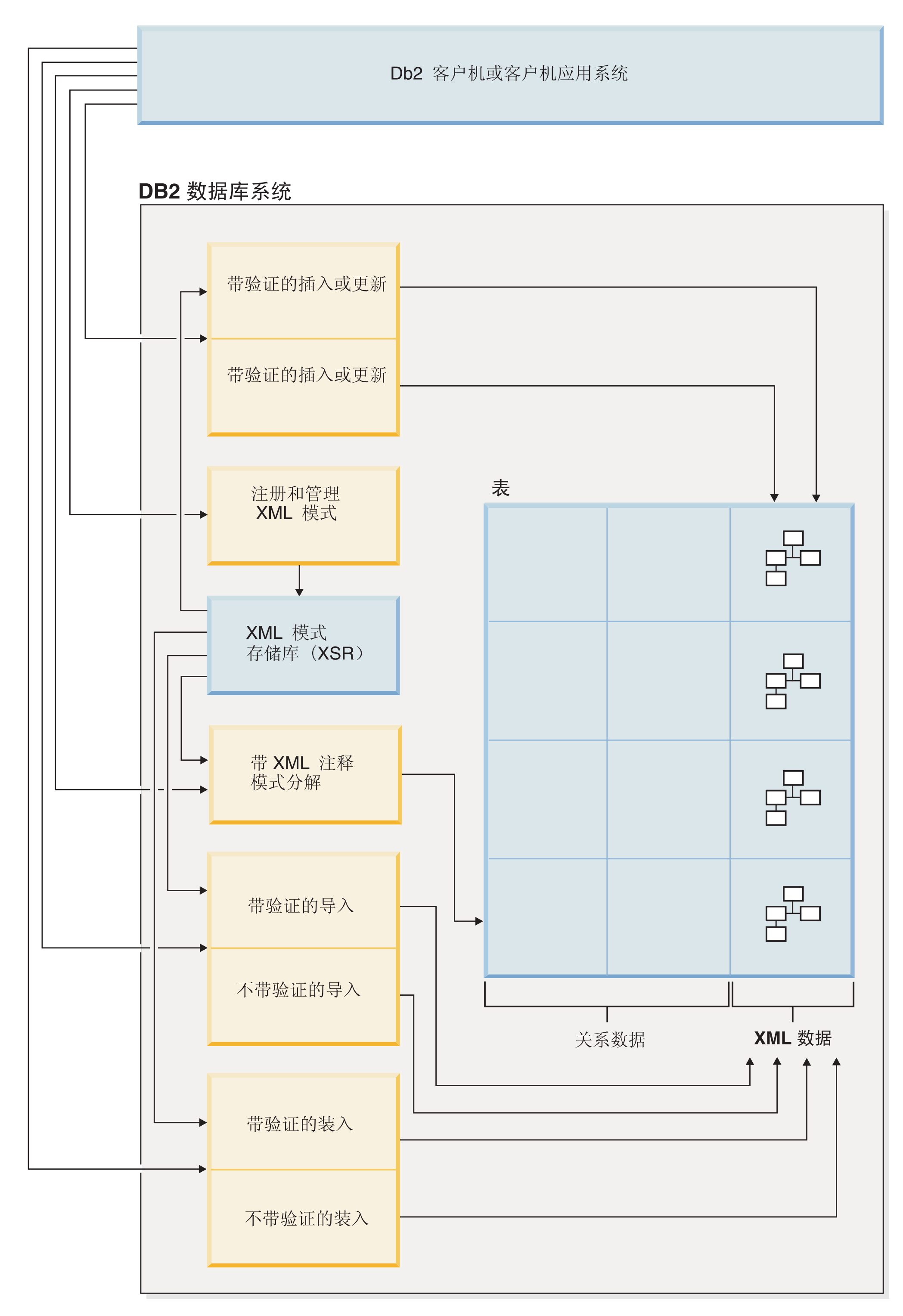


图 1. 用于输入 XML 数据的方法

使用的输入方法取决于要完成的任务:

插入或更新

可使用 INSERT SQL 语句将格式良好的文档插入到 XML 列中。如果能够成功分析文档，那么说明文档的格式良好。在执行插入或更新操作期间是否验证

XML 文档是可选的。如果执行验证，那么必须首先向 XML 模式存储库（XSR）注册 XML 模式。文档是使用 UPDATE SQL 语句或使用 XQuery 更新表达式更新的。

带注释的 XML 模式分解
可使用带注释的 XML 模式分解来分解 XML 文档中的数据，或者将该数据存储在关系列和 XML 列中。分解会根据添加至 XML 模式文档的注释将数据存储在列中。这些注释将 XML 文档中的数据映射至表列。

分解功能所引用的 XML 模式文档存储在 XML 模式存储库（XSR）中。

导入
可使用导入实用程序将 XML 文档导入到 XML 列中。在导入 XML 文档时是否进行验证是可选的。如果执行验证，那么必须首先向 XML 模式存储库（XSR）注册验证文档时所使用的 XML 模式。

XML 模式存储库（XSR）注册
XML 模式存储库（XSR）存储用于验证或分解 XML 文档的 XML 模式。要对依赖于这些模式的 XML 文档执行其他任务，通常需要先注册 XML 模式。XML 模式是使用存储过程或命令向 XSR 注册的。

输出

第 6 页的图 2 显示了可用于从数据库系统中检索 XML 数据的各种方法。

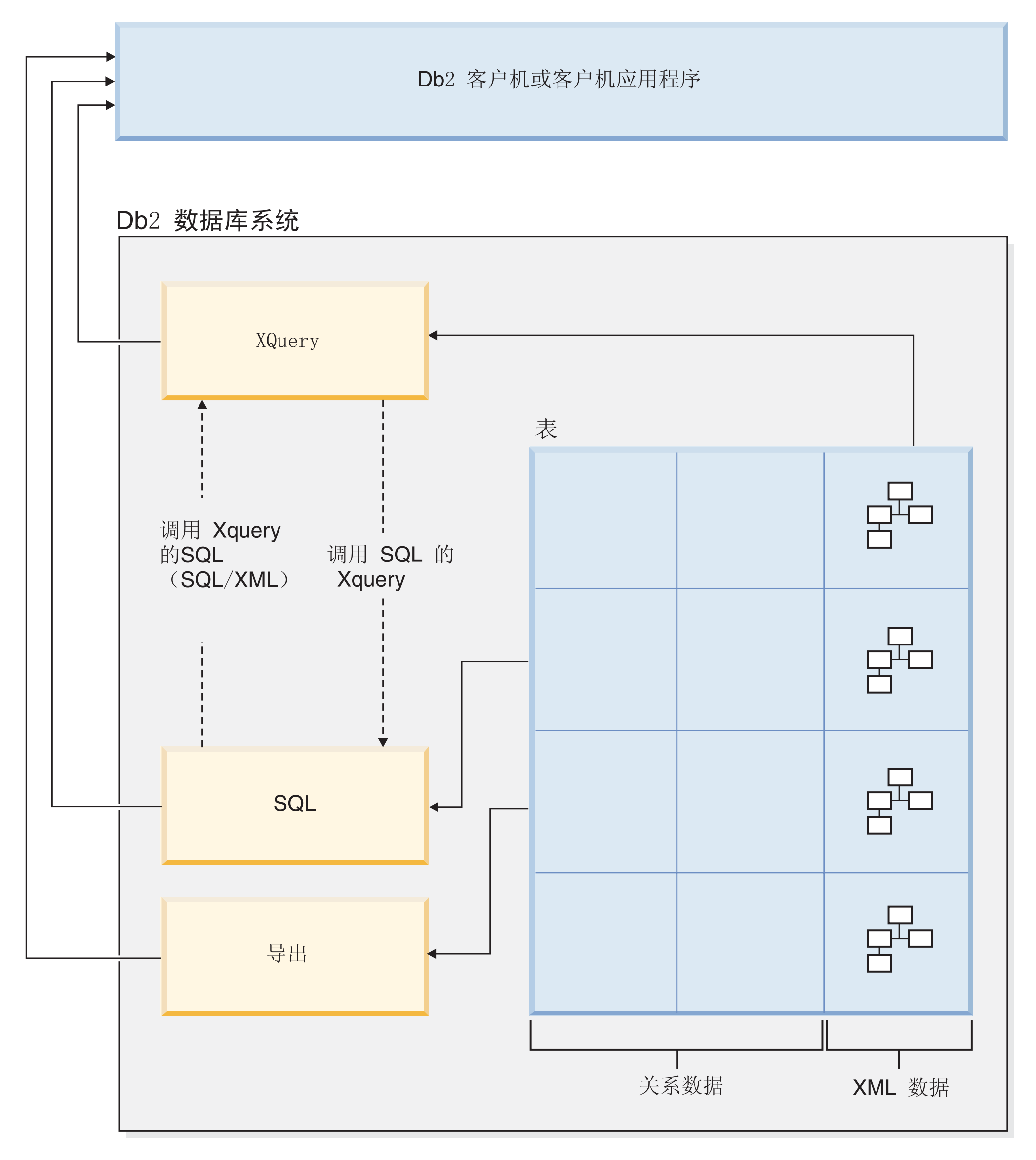


图 2. 用于输出 XML 数据的方法

使用的输出方法取决于要完成的任务:

XQuery

XQuery 是一种可用于在 XML 文档内进行查询的语言。它满足了查询结构变化非常大的 XML 数据的特定需求，与查询结构可预测的关系数据是不同的。

XQuery 可以自己调用自己，也可以通过 XQuery 函数 db2-fn:xmlcolumn 和 db2-fn:sqlquery 调用 SQL 来查询存储在 DB2 数据库中的 XML。db2-fn:xmlcolumn 将检索整个 XML 列，而 db2-fn:sqlquery 将检索基于 SQL 全查询的 XML 值。

SQL 当使用 SQL 全查询来查询 XML 数据时，将在列级别进行查询。因此，该查询只能返回整个 XML 文档；仅仅使用 SQL 是不可能返回 XML 文档包含的片段的。要在 XML 文档内进行查询，必须使用 XQuery。可以使用 SQL/XML 函数 XMLQUERY 或 XMLTABLE 或者使用 XMLEXISTS 谓词并通过调用 SQL 来调用 XQuery。XMLQUERY 函数将 XQuery 表达式的结果以 XML

序列形式返回。XMLTABLE 函数将 XQuery 表达式的结果以表的形式返回。XMLEXISTS SQL 谓词确定 XQuery 表达式是否会返回非空序列。

还可使用若干发布函数通过存储在 DB2 数据库服务器中的 XML 数据来构造 XML 值。使用这些发布函数构造的 XML 值不必是格式良好的 XML 文档。

导出
可使用导出实用程序从 XML 列中导出 XML 文档。已导出的 XML 数据与主数据文件中包含的已导出的关系数据存储在不同位置。有关每个已导出的 XML 文档的详细信息没有直接存储在已导出的主数据文件中。在主数据文件中，这些详细信息是由 XML 数据说明符（XDS）表示的。

比较 XML 模型和关系模型

设计数据库时，需要确定数据更适合 XML 模型还是关系模型。 设计可利用 DB2 数据库的混合特点，即，同时在一个数据库中支持关系数据和 XML 数据的能够。

虽然本讨论解释了这两个模型之间的主要区别以及适用于每个模型的因素，但还有很多因素可帮助您选择最合适的实现。将本讨论作为一个准则，了解可能会影响特定实现的所有因素。

XML 数据与关系数据之间的主要区别

XML 数据是分层数据；关系数据用逻辑关系模型表示
XML 文档以层次结构形式包含有关数据项之间的关系的信息。对于关系模型，可以定义的唯一关系类型是父表和从属表关系。

XML 数据能够自描述；而关系数据不能
XML 文档不仅包含数据，还包含有关用于说明数据的概念的标记。单个文档可以有不同类型的数据。对于关系模型，数据的内容由其列定义定义。列中的所有数据必须是相同类型的数据 。

XML 数据具有固定排序；而关系数据没有
对于 XML 文档，假定指定的数据项顺序是文档中数据的顺序。通常没有其他方法来指定文档内的顺序。对于关系数据，除非对一列或多列指定 ORDER BY 子句，否则不能保证行的顺序。

影响数据模型选择的因素

存储的数据种类可帮助您确定存储方式。例如，如果数据天生是分层且自描述的，那么可以将它存储为 XML 数据。但是，其他因素也可能会影响您决定要使用的模型。

需要最大灵活性时
关系表遵循非常严格的模型。例如，将一个表规范化为许多表或将许多表反向规范化为一个表可能非常困难。如果经常更改数据设计，那么将它表示为 XML 数据是较好的选择。例如，XML 模式可随时间演进。

数据检索需要最好性能时
序列化和解释 XML 数据会产生一些开销。如果性能比灵活性更重要，那么关系数据可能是较好的选择。

数据以后作为关系数据处理时
如果对数据的后续处理取决于要存储在关系数据库中的数据，那么使用分解将部分数据作为关系数据存储可能较合适。将联机分析处理（OLAP）应用于数

据仓库中的数据就是这样一个示例。此外，如果需要将对 XML 文档的其他处理作为一个整体，那么在这种情况下，较合适的方法是将部分数据作为关系数据存储并存储整个 XML 文档。

数据组件在层次结构外具有意义时

数据本身可能有固有的分层格式，但子组件不需要父组件提供值。例如，采购订单可能包含部件号。最好将带有部件号的采购订单以 XML 文档的形式表示。但是，每个部件号都有一个与它关联的部件描述。最好将部件描述包括在关系表中，因为部件号和部件描述之间的关系在逻辑上与使用这些部件号的采购订单无关。

数据属性适用于所有数据，或仅适用于一小部分数据时

一些数据集可能有大量属性，但只有一小部分属性适用于任何特定数据值。例如，在零售目录中，可能有许多数据属性，例如，大小、颜色、重量、材料、样式、织法、电源需求或燃料需求。对于该目录中的任何给定项，只有一部分属性相关：电源需求对于电锯有意义，但对于煤没有意义。很难用关系模型表示和搜索这种类型的数据，但使用 XML 模型来表示和搜索就相对要容易一些。

数据复杂性与容量之比较高时

在许多情况下，少量数据中包含了高度结构化的信息。使用关系模型表示该数据将涉及复杂的星型模式，在该模式中每个维表连接至许多个维表，并且其中大多数表只有少量行。表示此数据的一种较好方法是使用具有 XML 列的单个表，并且对该表创建视图，每个视图表示一个维。

需要引用完整性时

不能将 XML 列定义为引用约束的一部分。因此，如果 XML 文档中的值需要参与引用约束，那么应将数据作为关系数据存储。

需要经常更新数据时

只能通过替换整个文档来更新 XML 列中的 XML 数据。如果需要频繁更新非常大的文档中较小片段内包含的大量行，那么将数据存储在非 XML 列中可能会效率更高。但是，如果正在更新的文档很小并且一次只更新少量文档，那么作为 XML 数据存储效率也会很高。

XQuery 和 XPath 数据模型

XQuery 表达式对 XQuery 和 XPath 数据模型（XDM）的实例进行运算并返回数据模型的实例。XDM 是对一个或多个 XML 文档或片段的抽象表示。数据模型会定义 XQuery 中的表达式的允许值，包括中间计算期间使用的值。

将 XML 数据解析为 XDM，并在 XQuery 处理数据之前针对模式来验证这些数据。在生成数据模型期间，将解析输入 XML 文档，并将它转换为 XDM 的实例。在解析文档时，可以进行验证，也可以不进行验证。

XDM 是按照原子值和节点序列来进行描述的。

序列和项

XQuery 和 XPath 数据模型（XDM）的实例为序列。*序列*是 0 个项或多个项的有序集合。一个*项*就是一个原子值或一个节点。

一个序列可以包含节点、原子值或者是节点和原子值的任意组合。例如，下面列表中的每个条目都是一个序列:

- 36
- <dog/>
- (2, 3, 4)
- (36, <dog/>, "cat")
- ()

除列表中的条目之外，存储在 DB2 数据库的 XML 列中的 XML 文档是一个序列。

示例中用来表示序列的表示法，与用来构造 XQuery 中的序列的语法是一致的:

- 序列中的每项之间用逗号分隔。
- 整个序列是用圆括号括起来的。
- 一对空的圆括号表示一个空序列。
- 如果一个项在它自身上方出现，那么相当于一个只包含一项的序列。

  例如，序列 (36) 与原子值 36 没有区别。

不能对序列进行嵌套。当组合两个序列时，获得的结果始终是节点和原子值的平铺序列。例如，将序列 (2, 3) 追加至序列 (3, 5, 6) 时将生成单个序列 (3, 5, 6, 2, 3)。因为决不会出现嵌套序列，所以，组合这些序列时并不会生成 (3, 5, 6, (2, 3))。

不包含任何项的序列称为*空序列*。可使用空序列来表示缺少的信息或未知信息。

原子值

*原子值*是由 XML 模式定义的其中一种内置原子数据类型的实例。这些数据类型包括 String、Integer、Decimal、Date 和其他原子类型。这些类型都被描述为原子类型，原因是它们无法再细分了。

与节点不同的是，原子值没有标识。原子值的每个实例（例如，整数 7）与该值的其他每个实例都完全相同。

下列示例是一些生成原子值的方法:

- 通过一个称为“原子化”的过程从节点中抽取。每当需要原子值序列时，表达式就会使用原子化。
- 指定为数字或字符串文字。XQuery 会将文字解释为原子值。例如，下列文字就会被解释为原子值:
  - “this is a string”（类型为 xs:string）
  - 45（类型为 xs:integer）
  - 1.44（类型为 xs:decimal）
- 由构造函数计算获得。例如，以下构造函数将根据字符串“2005-01-01”来构建类型为 xs:date 的值:

  ```
  xs:date("2005-01-01")
  ```
- 由内置函数 fn:true() 和 fn:false() 返回。这些函数将返回布尔值 `true` 和 `false`。这些值不能表示为文字。
- 由多种表达式（例如，算术表达式和逻辑表达式）返回。

节点层次结构

组成一个或多个*层次结构*或*树*的节点序列，这些层次结构或树由一个根节点和可从该根节点直接或间接访问的所有节点组成。每个节点只属于一个层次结构，而每个层次结构只有一个根节点。DB2 支持以下 6 种节点：文档、元素、属性、文本、处理指示信息和注释。

以下 XML 文档 products.xml 包括一个根元素 products，该根元素又包含一些 product 元素。每个 product 元素都有一个名为 pid（产品标识）的属性以及一个名为 description 的子元素。description 元素包含名为 name 和 price 的子元素。

```
<products>
  <product pid="10">
    <description>
      <name>Fleece jacket</name>
      <price>19.99</price>
    </description>
  </product>
  <product pid="11">
    <description>
      <name>Nylon pants</name>
      <price>9.99</price>
    </description>
  </product>
</products>
```

第 11 页的图 3 显示 products.xml 的数据模型的简化表示。该图中包括文档节点（D）、元素节点（E）、属性节点（A）和文本节点（T）。

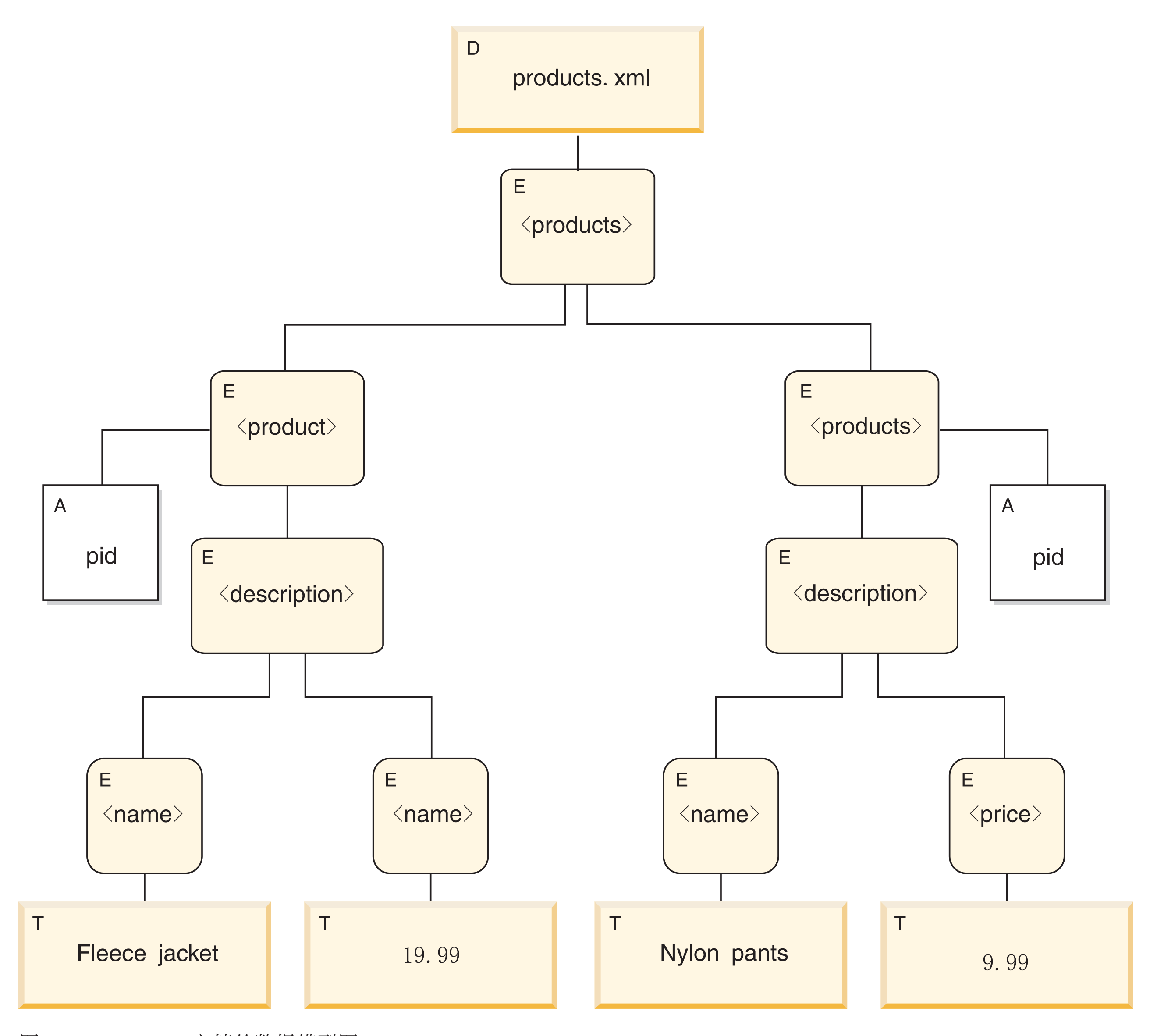


图 3. products.xml 文档的数据模型图

如示例中所示，一个节点可将其他节点作为子代，从而组成一个或多个*节点层次结构*。在该示例中，product 元素是 products 的子代。description 元素却是 product 的子代。name 和 price 元素都是 description 元素的子代。值为 Fleece Jacket 的文本节点是 name 元素的子代，而文本节点 19.99 是 price 元素的子代。

节点属性

每个节点都具有一些*特性*，这些特性用来描述该节点的特征。例如，节点的特性可能包括：节点的名称、子代、父代、属性以及用来描述该节点的其他信息。节点种类确定为特定节点提供了哪些属性。

一个节点可以具有下面的一个或多个属性:

node-name
节点的名称，表示为 QName。

父代　是当前节点的父代的节点。

type-name
: 节点的动态（运行时）类型（也称为*类型注释*）。

子代 是当前节点的子代的节点序列。

属性 属于当前节点的一组属性节点。

string-value
: 可以从节点中抽取的字符串值。

typed-value
: 可以从节点中抽取的由零个或多个原子值组成的序列。

名称空间作用域
: 与节点相关联的作用域内名称空间。

content
: 节点的内容。

节点种类

DB2 支持以下 6 种节点：文档、元素、属性、文本、处理指示信息和注释。

文档节点

文档节点包含 XML 文档。

文档节点可以具有多个子代，也可以没有子代。子代可以是元素节点、处理指令节点、注释节点和文本节点。

文档节点的字符串值等于将它的所有后代文本节点的内容按文档顺序进行并置的结果。字符串值的类型为 xs:string。文档节点的类型值与字符串值相同，但类型值的类型为 xdt:untypedAtomic。

文档节点具有下列节点属性：

- children（可能是空的）
- string-value
- typed-value

可以在 XQuery 表达式中使用经过计算获得的构造函数来构造文档节点。db2-fn:xmlcolumn 函数还可以返回一系列文档节点。

元素节点

元素节点包含 XML 元素。

一个元素可以具有一个父代，也可以没有父代；同时，它可以具有多个子代，也可以没有子代。子代可以是元素节点、处理指令节点、注释节点和文本节点。文档节点和属性节点决不会是元素节点的子代。但是，可以认为元素节点是它自己的属性的父代。元素节点的属性必须具有唯一的 QName。

元素节点具有下列节点属性：

- node-name
- parent（可能是空的）
- type-name

- children（可能是空的）
- attributes（可能是空的）
- string-value
- typed-value
- in-scope-namespaces

可以在 XQuery 表达式中使用直接构造函数或者经过计算获得的构造函数来构造元素节点。

元素节点的 type-name 属性指示它的类型值与字符串值之间的关系。例如，如果一个元素节点具有 type-name 属性 xs:decimal 和字符串值“47.5”，那么类型值就是小数值 47.5。如果元素节点的 type-name 属性是 xdt:untyped，那么元素的类型值等于其字符串值，并且类型为 xdt:untypedAtomic。

属性节点

属性节点表示 XML 属性。

属性节点可以具有一个父代，也可以没有父代。可将拥有属性的元素节点认为是它的父代，尽管属性节点不是它的父元素的子代。

属性节点具有下列节点属性:

- node-name
- parent（可能是空的）
- type-name
- string-value
- typed-value

可以在 XQuery 表达式中使用直接构造函数或者经过计算获得的构造函数来构造属性节点。

属性节点的 type-name 属性指示它的类型值与字符串值之间的关系。例如，如果一个属性节点具有 type-name 属性 xs:decimal 和字符串值“47.5”，那么它的类型值就是小数值 47.5。

文本节点

文本节点包含 XML 字符内容。

文本节点可以具有一个父代，也可以没有父代。作为文档节点或元素节点的子代的文本节点决不会作为相邻同代出现。当构造文档节点或元素节点时，任何相邻的文本节点同代都会被合并成单个文本节点。如果获得的文本节点是空的，那么会将它废弃。

文本节点具有下列节点属性:

- content（可能是空的）
- parent（可能是空的）

可以在 XQuery 表达式中使用经过计算获得的构造函数来构造文本节点，也可以通过直接元素构造函数的操作来构造文本节点。

处理指令节点

处理指令节点会封装 XML 处理指令。

处理指令节点可以具有一个父代，也可以没有父代。处理指令的内容不能包含字符串 ?>。处理指令的目标必须是一个 NCName。该目标用来标识要将指示信息发送给的应用程序。

处理指令节点具有下列节点属性:

- target
- content
- parent（可能是空的）

可以在 XQuery 表达式中使用直接构造函数或者经过计算获得的构造函数来构造处理指令节点。

注释节点

注释节点包含 XML 注释。

注释节点可以具有一个父代，也可以没有父代。注释节点的内容不能包括字符串"--"（两个连字符），并且最后一个字符不能是连字符（-）。

注释节点具有下列节点属性:

- content
- parent（可能是空的）

可以在 XQuery 表达式中使用直接构造函数或者经过计算获得的构造函数来构造注释节点。

节点的文档顺序

一个层次结构中的所有节点都要遵从某一顺序（即，*文档顺序*）。按照该顺序，每个节点都将在其子代前面出现。如果节点层次结构是用已序列化的 XML 表示的，那么文档顺序与节点的出现顺序相对应。

层次结构中的节点按以下顺序出现:

- 根节点是第一个节点。
- 元素节点在它们的子代前面出现。
- 属性节点紧跟在与它们相关联的元素节点后面出现。属性节点的相对顺序可以是任意的，但是在处理查询期间此顺序不会改变。
- 同代的相对顺序由它们在节点层次结构中的顺序来确定。
- 一个节点的子代和后代将在该节点后面的同代前面出现。

节点标识

每个节点都有一个唯一标识。即使两个节点的名称和值都相同，也可以将它们区分开。然而，原子值没有标识。

节点标识与 ID-type 属性不相同。XML 文档中的元素可由文档作者给定 ID-type 属性。然而，节点标识是由系统自动为每个节点指定的，用户无法直接看见节点标识。

节点标识用于处理下列类型的表达式:

- 节点比较。**is** 运算符使用节点标识来确定两个节点是否具有相同标识。
- 路径表达式。路径表达式使用节点标识来消除重复的节点。
- 序列表达式。**union**、**intersect** 或 **except** 运算符使用节点标识来消除重复的节点。

节点的类型值和字符串值

每个节点都同时有*类型值*和*字符串值*。这两个节点属性用于某些 XQuery 操作（例如，原子化）和函数（例如，fn:data、fn:string 和 fn:deep-equal）的定义中。

表 1. 节点的字符串值和类型值

节点种类	**字符串值**	**类型值**
文档	xs:string 数据类型的实例，它是将它的所有后代文本节点的内容按文档顺序进行并置的结果。	xdt:untypedAtomic 数据类型的实例，它是将它的所有后代文本节点的内容按文档顺序进行并置的结果。
XML 文档中的元素	xs:string 数据类型的实例，它是将它的所有文本节点后代的内容按文档顺序进行并置的结果。	xdt:untypedAtomic 数据类型的实例，它是将它的所有文本节点后代的内容按文档顺序进行并置的结果。
XML 文档中的属性	xs:string 数据类型的实例，它表示原始 XML 文档中的属性值。	xdt:untypedAtomic 数据类型的实例，它表示原始 XML 文档中的属性值。
文本	作为 xs:string 数据类型的实例的内容。	作为 xdt:untypedAtomic 数据类型的实例的内容。
注释	作为 xs:string 数据类型的实例的内容。	作为 xs:string 数据类型的实例的内容。
处理指令	作为 xs:string 数据类型的实例的内容。	作为 xs:string 数据类型的实例的内容。

支持 XML 的工具

IBM 和第三方工具支持使用 pureXML 功能部件。下列工具是随 DB2 数据库服务器提供的或者是可单独从 IBM 下载的:

IBM Data Studio: 对 XML 的支持包括以下几项:

- **存储过程**: 可以创建和运行包含 XML 数据类型作为输入和输出参数的存储过程。
- **数据输出**: 可以树型或文本形式查看包含在 XML 列中的文档。
- **SQL 编辑器**: 可创建同时使用关系数据和 XML 数据的 SQL 语句和 XQuery 表达式。
- **XML 模式**: 可以管理 XML 模式存储库（XSR）中的模式文档，包括注册和删除模式以及编辑模式文档。
- **XML 文档验证**: 可以针对在 XSR 中注册的模式验证 XML 文档。
- **用户定义的 SQL 函数**: 可创建并运行使用 XML 参数的用户定义的 SQL 函数。

DB2 控制中心: DB2 控制中心支持对它的许多管理功能使用本机 XML 数据类型。此更新使数据库管理员能够在一个 GUI 工具中处理 XML 数据和关系数据。

受支持的管理任务的示例包括:

- 创建带有 XML 列的表
- 使用新的“创建索引”向导来创建 XML 列索引
- 查看存储在 XML 列中的 XML 文档的内容
- 使用处理 XML 文档所需的 XML 模式、DTD 和外部实体
- 收集包含了 XML 列的表的统计信息

命令行处理器: 若干个 DB2 命令支持本机存储 XML 数据。可以从 DB2 命令行处理器（CLP）使用关系数据旁边的 XML 数据。可以从 CLP 执行的任务的示例包括:

- 通过对 XQuery 语句加上 XQUERY 关键字前缀来发出 XQuery 语句。
- 导入和导出 XML 数据。
- 对 XML 列收集统计信息。
- 使用 XML 数据类型的 IN、OUT 或 INOUT 参数来调用存储过程。
- 使用处理 XML 文档所需的 XML 模式、DTD 和外部实体
- 重组基于 XML 数据的索引和包含 XML 列的表。
- 分解 XML 文档。

DB2 Development Add-In for Microsoft Visual Studio .NET: 可以使用 Development Add-In for Microsoft Visual Studio .NET 来创建具有 XML 列的表和基于 XML 数据的索引。在此工具中像创建任何其他列一样创建 XML 列。只需要将数据类型指定为 XML。可以通过使用此工具中的“XML 索引设计器”来创建索引。不必如使用 CREATE INDEX 语法创建基于 XML 数据的索引时所要求的那样手动指定 XML 模式表达式。相反，可以从已注册的 XML 模式的树型表示、XML 列中的文档或本地文件中的 XML 模式以图形方式选择想要建立索引的 XML 节点。该工具将为您生成 XML 模式表达式。或者，可以手动指定 XML 模式表达式。指定了所有其他索引属性之后，该工具将为您生成索引。

EXPLAIN: 可以在 XQuery 语句和 SQL/XML 语句中发出 EXPLAIN 语句，以便快速查看这些语句的访问方案，包括 DB2 数据库服务器是否使用索引。要发出 XQuery 语句的 EXPLAIN 语句，可使用 XQuery 关键字，后跟用单引号或双引号引起来的 XQuery 语句，如以下示例所示:

```
EXPLAIN PLAN SELECTION FOR XQUERY 'for $c in
db2-fn:xmlcolumn("XISCANTABLE.XMLCOL" )/a[@x="1"]/b[@y="2"] return $c'
```

DB2 将捕获 EXPLAIN 表中的访问方案信息。所有 XML 列的期望序列大小都存储在 EXPLAIN_STREAM 表的 SEQUENCE_SIZES 列中。您可能还会注意到 EXPLAIN_PREDICATE 表中存在某些数据，它们是一些您无法识别的谓词。这些谓词由 DB2 数据库服务器在 EXPLAIN 操作期间对索引扫描中使用的 XPath 表达式进行求值时生成。您不需要评估此谓词信息。这些谓词不是优化器方案的一部分，因此在 PREDICATE_ID 和 FILTER_FACTOR 列中的值为 -1。

或者，通过使用 Visual Explain 工具来查看这些访问方案的图形描述，可以避免手动解释 EXPLAIN 表。下列节点显示在图形中以说明 XML 操作:

IXAND
: 指示 DB2 数据库服务器已将 AND 谓词应用于多个索引扫描的结果。

XISCAN
: 指示 DB2 数据库服务器已使用基于 XML 数据的索引来访问数据。

XSCAN
: 指示 DB2 数据库服务器已对 XPath 表达式求值，并且已从 XML 文档中抽取 XML 文档片段。

XANDOR
: 指示 DB2 数据库服务器已将 AND 和 OR 谓词应用于多个索引扫描的结果。

XTQ

指示 DB2 数据库服务器使用了特殊表队列（TQ）将全局 XML 序列中的每项发送到其原始数据库分区，根据该项的评估从 XML 文档检索 XML 数据，然后将 XML 数据聚集到输出序列中。

pureXML 的联合支持

在联合环境中，可使用包含 XML 列中存储的 XML 文档的远程数据源。可查询并处理远程 XML 数据，包括将 XML 文档分解为远程表。

需要先为包含 XML 列（其中存储了要使用的文档）的远程表创建昵称，才能使用远程 XML 数据。

有关设置包括 XML 数据源的联合系统的更多信息，请参阅联合服务器文档中的“使用远程 XML 数据”。

pureXML 的复制和事件发布支持

XML 数据类型的 WebSphere® Replication Server 和 WebSphere® Data Event Publisher 支持允许您复制并发布存储在 XML 列中的 XML 文档。

可使用 Q 复制在数据库之间复制 XML 文档，也可使用事件发布将文档复制至应用程序。

有关为包括存储在 XML 列中的 XML 文档的数据库设置 Q 复制和事件发布的更多信息，请参阅 WebSphere Replication Server 和 WebSphere Data Event Publisher 文档中的“XML 数据类型”及父主题。

关于 XML 支持的文章

通过 developerWorks® 信息管理可获得关于利用 XML 支持的其他文章。这些文章涉及广泛的主题，包括迁移和数据移动、通用概述、逐步教程和使用 XML 数据的最佳做法。

可在下面的网址中找到这些文章：www.ibm.com/developerworks/db2/zones/xml/。

注： developerWorks 并非 DB2 信息中心的一部分。此链接将打开 DB2 信息中心以外的内容。

第 2 章 pureXML 教程

可使用 pureXML XML 数据类型来定义表列，以便您可在每行中存储单个格式良好的 XML 文档。本教程说明如何设置 DB2 数据库以存储 XML 数据以及如何使用 pureXML 功能部件执行基本操作。

在完成本教程之后，您将能够执行下列任务:

- 创建可存储 XML 数据的 DB2 数据库和表
- 创建基于 XML 数据的索引
- 将 XML 文档插入到 XML 类型的列中
- 更新存储在 XML 列中的 XML 文档
- 根据 XML 文档的内容删除行
- 查询 XML 数据
- 针对 XML 模式验证 XML 文档
- 使用 XSLT 样式表变换 XML 文档

C++、Java 和 PHP 之类的应用程序编程语言支持 XML 数据类型。可以编写应用程序来将 XML 数据存储在 DB2 数据库表中，从表中检索数据或调用具有 XML 参数的存储过程或用户定义的函数。

本教程是为单一分区数据库环境编写的，但您也可在分区数据库环境中使用 pureXML 功能部件。

先决条件

在 DB2 命令窗口中，通过发出 `db2 -td~` 命令（带有 **-td~** 选项的 db2 命令）来启动 DB2 命令行处理器。[1]

-td 选项会将颚化音符号（~）设置为语句终止字符。指定缺省分号（**-t** 选项）以外的终止字符可确保不会错误解释使用名称空间声明的语句或查询，原因是名称空间声明也是用分号终止的。本教程中的示例使用 ~ 终止字符。

可将课程中的示例以交互方式输入或复制并粘贴到 DB2 命令行处理器中。

或者，可使用命令编辑器来以交互方式发出命令和语句。

名称空间: 教程中使用的 XML 文档包含名称空间。使用包含名称空间的 XML 文档时，指定名称空间的所有查询和关联操作（如通过使用 CREATE INDEX 语句来创建基于 XML 数据的索引或使用 XQuery 表达式来查询 XML 数据）必须声明同一名称空间才能生成期望的结果。使用包含名称空间的 XML 文档时，声明名称空间是标准名称空间行为。

1. 在 Windows 操作系统上，先使用 db2cmd 命令启动 DB2 命令窗口，然后发出 db2 -td~ 命令。

课程 1: 创建可以存储 XML 数据的 DB2 数据库和表

本课程描述如何使用包含 XML 列的表创建数据库。

在本教程中，您将 XML 数据存储在一个表中，而此表包含具有 XML 类型的列的表。请遵循下列步骤来创建本教程中使用的数据库和表:

1. 通过发出以下命令来创建称为 XMLTUT 的数据库:

   ```
   CREATE DATABASE xmltut~
   ```

   缺省情况下，将在创建数据库时启用自动存储器。自动存储器可以提高性能并且更容易管理 XML 数据，因为它将生成可以根据需要进行扩展的数据库管理空间（DMS）表空间。

   而且，在缺省情况下，数据库使用 UTF-8（Unicode）代码集。如果选择将 XML 数据存储在使用 UTF-8 以外的代码集的数据库中，那么最好以不进行代码页转换的形式（例如 BIT DATA、BLOB 或 XML）插入此数据。要阻止在执行 XML 解析期间使用字符数据类型，从而防止可能发生的字符替换，请将 **ENABLE_XMLCHAR** 配置参数设置为 **NO**。
2. 与数据库连接:

   ```
   CONNECT TO xmltut~
   ```
3. 创建一个名为 Customer 并且包含称为 INFO 的 XML 列的表:

   ```
   CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY, Info XML)~
   ```

   存储 XML 数据或者对 XML 数据建立索引时不需要使用主键。

   还可以使用 ALTER TABLE SQL 语句将一个或多个 XML 列添加至表。

返回至教程

课程 2: 创建基于 XML 数据的索引

本课程描述如何创建 XML 数据索引。基于 XML 数据的索引可以提高查询 XML 列的性能。请对您在谓词中以及跨文档进行连接时频繁使用的 XML 元素或属性建立索引。

关系索引和 XML 数据索引都对列建立索引。但是，关系索引是对整个列建立索引，而 XML 数据索引只是对列的某部分建立索引。通过指定 XML 模式来指明要对 XML 列的哪些部分建立索引，该模式是受限制的 XPath 表达式。还必须指定已建立索引的值将采用哪种数据类型进行存储。通常，您选择的类型应该与查询中使用的类型相同。

与关系索引一样，建议您对谓词和交叉文档连接中频繁使用的 XML 元素或属性建立索引。

只能对单个 XML 列建立索引；不支持组合索引。但是，一个 XML 列可以有多个索引。

并不是 CREATE INDEX 语句的所有子句都适用于基于 XML 数据的索引。有关详细信息，请参阅 CREATE INDEX 语句。

要创建 XML 数据索引，请发出以下语句:

```
CREATE INDEX cust_cid_xmlidx ON Customer(Info)
GENERATE KEY USING XMLPATTERN
  'declare default element namespace "http://posample.org"; /customerinfo/@Cid'
AS SQL DOUBLE~
```

此语句将对 CUSTOMER 表的 INFO 列中 <customerinfo> 元素的 Cid 属性值建立索引。缺省情况下，对 XML 数据建立索引之后，如果未能将此 XML 数据转换为指定的数据类型 SQL DOUBLE，那么不会创建索引条目，也不会返回错误。

您指定的 XML 模式区分大小写。例如，如果 XML 文档中包含 cid 属性而不是 Cid 属性，那么这些文档与此索引将不匹配。

返回至教程

课程 3：将 XML 文档插入到 XML 类型列中

可使用 INSERT SQL 语句将格式良好的 XML 文档插入到 XML 类型的列中。本课程描述如何使用 INSERT SQL 语句将格式良好的 XML 文档插入到 XML 列中。

本课程描述如何使用命令行处理器将 XML 文档手动插入到 XML 类型列中。但是，通常会使用应用程序来插入 XML 文档。

尽管可以通过使用 XML 类型、二进制类型或字符类型来插入 XML 数据，但是为了避免发生代码页转换问题，请使用 XML 类型或二进制类型。在本课程中，XML 文档是字符文字。在大多数情况下，不能直接对具有 XML 数据类型的目标指定字符串数据；您必须首先使用 XMLPARSE 函数来显式解析数据。但是，在 INSERT、UPDATE 或 DELETE 操作中，可以直接对 XML 列指定字符串数据，而不需要显式调用 XMLPARSE 函数。在这三种情况下，将隐式解析字符串数据。有关更多信息，请参阅 XML 解析文档。

要将三个 XML 文档插入到您在课程 1 中创建的 Customer 表，请发出下列语句：

```
INSERT INTO Customer (Cid, Info) VALUES (1000,
'<customerinfo xmlns="http://posample.org" Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>')~

INSERT INTO Customer (Cid, Info) VALUES (1002,
'<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>')~

INSERT INTO Customer (Cid, Info) VALUES (1003,
'<customerinfo xmlns="http://posample.org" Cid="1003">
  <name>Robert Shoemaker</name>
```

```
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-2937</phone>
</customerinfo>')~
```

要确认是否已成功插入记录，请发出以下语句:

```
SELECT * from Customer~
```

返回至教程

课程 4: 更新存储在 XML 列中的 XML 文档

本课程描述如何通过将 UPDATE SQL 语句与 XQuery 更新表达式配合使用或者单独使用 UPDATE SQL 语句来更新 XML 文档。

不使用 XQuery 更新表达式进行更新

如果您使用 UPDATE 语句而不使用 XQuery 更新表达式，那么必须执行全文档更新。

要更新您在课程 3 中插入的其中一个文档的 <street>、<city> 和 <pcode-zip> 元素的值，请发出以下语句:

```
UPDATE customer SET info =
'<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>1150 Maple Drive</street>
    <city>Newtown</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>Z9Z 2P2</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>'
WHERE XMLEXISTS (
 'declare default element namespace "http://posample.org";
  $doc/customerinfo[@Cid = 1002]'
  passing INFO as "doc")~
```

XMLEXISTS 谓词确保仅替换包含属性 Cid="1002" 的文档。请注意，XMLEXISTS 中的谓词表达式 [@Cid = 1002] 未指定为字符串比较 [@Cid = "1002"] 的原因。原因在于您在练习 2 中为 Cid 属性创建索引时使用了 DOUBLE 数据类型。为了使该索引与此查询匹配，在谓词表达式中不能将 Cid 指定为字符串。

要确认是否已更新 XML 文档，请发出以下语句:

```
SELECT * from Customer~
```

包含 Cid="1002" 的记录中包含已更改的 <street>、<city> 和 <pcode-zip> 值。

如果可以通过使用一个表的非 XML 列值来标识此表中的 XML 文档，那么可以使用 SQL 比较谓词来标识要更新的行。在前一个示例中，XML 文档中的 Cid 值还存储在 CUSTOMER 表的 CID 列中，您可能已使用 CID 列中的 SQL 比较谓词来标识行。在前一示例中，可以将 WHERE 子句替换为以下子句:

```
WHERE Cid=1002
```

使用 XQuery 更新表达式进行更新

如果您将 UPDATE 语句与 XQuery 更新表达式配合使用，那么可以更新现有 XML 文档的某些部分。

要更新现有 XML 文档中的客户地址，请发出以下 SQL 语句，它使用了 XQuery 变换表达式:

```
UPDATE Customer set Info =
  XMLQUERY(' declare default element namespace "http://posample.org";
  transform
  copy $mycust := $cust
  modify
    do replace $mycust/customerinfo/addr with
      <addr country="Canada">
        <street>25 EastCreek</street>
        <city>Markham</city>
        <prov-state>Ontario</prov-state>
        <pcode-zip>N9C 3T6</pcode-zip>
     </addr>
  return $mycust'
  passing INFO as "cust")
WHERE CID = 1002~
```

要更新客户地址，XMLQUERY 函数将执行其使用替换表达式的 XQuery 变换表达式，然后将已更新的信息返回至 UPDATE 语句，如下所示:

- XMLQUERY 传递子句使用标识 `cust` 以将 XML 列 INFO 中的客户信息传递至 XQuery 表达式。
- 在变换表达式的 **copy** 子句中，将获取客户信息的逻辑快照并将其指定给 `$mycust` 变量。
- 在变换表达式的 **modify** 子句中，替换表达式会替换客户信息副本中的地址信息。
- XMLQUERY 会在 `$mycust` 变量中返回已更新的客户文档。

要确认 XML 文档是否包含已更新的客户地址，请发出以下语句:

```
SELECT Info FROM Customer WHERE Cid = 1002~
```

返回至教程

课程 5: 删除 XML 数据

本课程描述如何使用 SQL 语句来删除整个 XML 文档或者仅删除 XML 文档的某些部分。

删除整个 XML 文档

要删除整个 XML 文档，可使用 DELETE SQL 语句。使用 XMLEXISTS 谓词来标识要删除的特定文档。

要从 INFO 列中仅删除其 <customerinfo> 元素具有 Cid=1003 属性的那些 XML 文档，请发出以下语句:

```
DELETE FROM Customer
WHERE XMLEXISTS (
 'declare default element namespace "http://posample.org";
  $doc/customerinfo[@Cid = 1003]'
  passing INFO as "doc")~
```

如果可以通过使用一个表的非 XML 列值来标识此表中的 XML 文档，那么可以使用 SQL 比较谓词来标识要删除的行。在前一个示例中，XML 文档中的 Cid 值还存储在 CUSTOMER 表的 CID 列中，您可能已使用以下 DELETE 语句执行了相同的操作，这会将 SQL 比较谓词应用于 CID 列以标识行：

```
DELETE FROM Customer WHERE Cid=1003~
```

要确认是否已删除 XML 文档，请发出以下 SQL 语句：

```
SELECT * FROM Customer~
```

将返回两条记录。

删除 XML 文档的某些部分

要仅删除 XML 文档的某些部分而不是删除整个文档，请使用包含"删除 XQuery 更新"表达式的 UPDATE SQL 语句。

要从 Cid 的值为 1002 的客户记录中删除所有电话信息，请发出以下使用 XMLQUERY 函数的 SQL 语句：

```
UPDATE Customer
SET info = XMLQUERY(
   'declare default element namespace "http://posample.org";
    transform
    copy $newinfo := $info
    modify do delete ($newinfo/customerinfo/phone)
    return $newinfo' passing info as "info")
WHERE cid = 1002~
```

要除去 <phone> 元素，XMLQUERY 函数将执行其使用删除表达式的 XQuery 变换表达式，然后将已更新的信息返回至 UPDATE 语句，如下所示：

- XMLQUERY 传递子句使用标识 `info` 将 XML 列 INFO 中的客户信息传递至 XQuery 表达式。
- 在变换表达式的 **copy** 子句中，将获取客户信息的逻辑快照并将其指定给 `$newinfo` 变量。
- 在变换表达式的 **modify** 子句中，删除表达式会删除客户信息副本中的 <phone> 元素。
- XMLQUERY 会在 `$newinfo` 变量中返回已更新的客户文档。

要确认客户记录中是否已不再包含 <phone> 元素，请发出以下语句：

```
SELECT * FROM Customer WHERE Cid=1002~
```

返回至教程

课程 6：查询 XML 数据

本课程描述如何使用 SQL 和/或 XQuery（使用 XQuery 表达式）来查询 XML 数据。

如果您仅使用 SQL，那么只能在列级别进行查询。也就是说，可以返回存储在列中的整个 XML 文档，但不能在文档内进行查询或者返回文档片段。要在 XML 文档内查询值或者返回文档片段，必须使用 XQuery。

本课程中的查询在 SQL 上下文中使用 XQuery，在 XQuery 上下文中使用 SQL。

重要事项: XQuery 区分大小写，但 SQL 不区分大小写。因此，在使用 XQuery 时，指定诸如表名和 SQL 模式名（缺省情况下，这两个名称都是大写）之类的名称时一定要小心。即使在 SQL 上下文中，XQuery 表达式仍将区分大小写。

在 SQL 上下文中查询

检索整个 XML 文档

要检索存储在名为 INFO 的列中的所有 XML 文档以及 CID 主键列中的值，请发出以下 SELECT 语句:

```
SELECT cid, info FROM customer~
```

此查询返回两个存储的 XML 文档。

检索和过滤 XML 值

要在 INFO 列的 XML 文档中查询，请发出以下 SELECT 语句，它将使用 XMLQUERY 函数来调用 XQuery 表达式:

```
SELECT XMLQUERY (
    'declare default element namespace "http://posample.org";
      for $d in $doc/customerinfo
   return <out>{$d/name}</out>'
      passing INFO as "doc")
FROM Customer as c
WHERE XMLEXISTS ('declare default element namespace "http://posample.org";
   $i/customerinfo/addr[city="Toronto"]' passing c.INFO as "i")~
```

在 XMLQUERY 函数中，首先指定缺省名称空间。此名称空间与先前插入的文档的名称空间匹配。**for** 子句指定通过 INFO 列中每个文档的 <customerinfo> 元素进行迭代。INFO 列是使用 **passing** 子句指定的，该子句将 INFO 列绑定至 **for** 子句中所引用的 doc 变量。然后，**return** 子句构造一个 <out> 元素，该元素包含 **for** 子句每次迭代生成的 <name> 元素。

WHERE 子句使用 XMLEXISTS 谓词来仅考虑 INFO 列中的一部分文档。此过滤仅生成 <city> 元素（沿指定的路径）的值为 Toronto 的那些文档。

此 SELECT 语句将返回以下已构造元素:

```
<out xmlns="http://posample.org"><name>Kathy Smith</name></out>
```

使用 db2-fn:sqlquery 时附带参数

要将值传递至 db2-fn:sqlquery 函数中的 SQL 全查询，请运行以下查询:

```
VALUES XMLQUERY (
    'declare default element namespace "http://posample.org";
  for $d in db2-fn:sqlquery(
    ''SELECT INFO FROM CUSTOMER WHERE Cid = parameter(1)'',
    $testval)/customerinfo
  return <out>{$d/name}</out>'
  passing 1000 as "testval" )~
```

XMLQUERY 函数通过使用标识 `testval` 将值 1000 传递至 XQuery 表达式。然后 XQuery 表达式通过使用 PARAMETER 标量函数将该值传递至 db2-fn:sqlquery 函数。

XQuery 表达式将返回以下已构造元素:

```
<out xmlns="http://posample.org">
   <name>Kathy Smith</name>
</out>
```

在 XQuery 上下文中查询

DB2 XQuery 特地提供了以下两个内置函数，以与 DB2 数据库配合使用：db2-fn:sqlquery 和 db2-fn:xmlcolumn。db2-fn:sqlquery 检索作为 SQL 全查询的结果表的序列。db2-fn:xmlcolumn 从 XML 列中检索序列。

如果查询直接调用 XQuery 表达式，那么必须在它前面添加不区分大小写的关键字 XQUERY。

注: 可以设置几个选项来定制命令行处理器环境，特别是用于显示 XQuery 表达式的结果。例如，按如下所示设置 **-i** 选项，以便更容易阅读 XQuery 表达式的结果:

```
UPDATE COMMAND OPTIONS USING i ON~
```

检索整个 XML 文档

要检索先前插入到 INFO 列中的所有 XML 文档，可以将 XQuery 与 db2-fn:xmlcolumn 或 db2-fn:sqlquery 配合使用。

使用 db2-fn:xmlcolumn

要检索 INFO 列中的所有 XML 文档，请运行以下查询:

```
XQUERY db2-fn:xmlcolumn ('CUSTOMER.INFO')~
```

缺省情况下，SQL 语句中的名称将自动转换为大写。因此，当使用 CREATE TABLE SQL 语句创建了 CUSTOMER 表时，表名和列名都为大写。因为 XQuery 区分大小写，所以在使用 db2-fn:xmlcolumn 指定表名和列名时必须使用正确的大小写。

此查询等价于 SQL 查询 `SELECT Info FROM Customer`。

使用 db2-fn:sqlquery

要检索 INFO 列中的所有 XML 文档，请运行以下查询:

```
XQUERY db2-fn:sqlquery ('SELECT Info FROM Customer')~
```

您不必采用大写字母来指定 INFO 名称和 CUSTOMER 名称，这是因为 SELECT 语句是在 SQL 上下文中处理的，因此不区分大小写。

检索部分 XML 文档

除了检索整个 XML 文档之外，还可以通过将 XQuery 与 db2-fn:xmlcolumn 或 db2-fn:sqlquery 配合使用来检索文档片段并过滤文档中存在的值。

使用 db2-fn:xmlcolumn

要返回包含 INFO 列中所有文档的 <name> 节点的元素，这些文档满足"<city> 元素（沿指定的路径）的值为 Toronto，请运行以下查询:

```
XQUERY declare default element namespace "http://posample.org";
  for $d in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo
  where $d/addr/city="Toronto"
  return <out>{$d/name}</out>~
```

db2-fn:xmlcolumn 函数从 CUSTOMER 表的 INFO 列中检索序列。**for** 子句将 `$d` 变量绑定至 CUSTOMER.INFO 列中的每个 <customerinfo> 元素。**where** 子句将文档限制为 <city> 元素（沿指定的路径）的值为 Toronto 的那些文档。**return** 子句将构造所返回的 XML 值。此值是一个 <out> 元素，它包含满足在 **where** 子句中所指定条件的所有文档的 <name> 元素，如下所示:

```
<out xmlns="http://posample.org">
<name>
              Kathy Smith
</name>
</out>
```

使用 **db2-fn:sqlquery**

要在 XQuery 表达式中发出全查询，请运行以下查询:

```
XQUERY declare default element namespace "http://posample.org";
  for $d in db2-fn:sqlquery(
    'SELECT INFO
     FROM CUSTOMER
     WHERE Cid < 2000')/customerinfo
  where $d/addr/city="Toronto"
  return <out>{$d/name}</out>~
```

在此示例中，首先在全查询中用非 XML CID 列中的特定值限制要被查询的 XML 文档集。此示例说明了 db2-fn:sqlquery 的优点：它允许在 XQuery 表达式中应用 SQL 谓词。然后，在 XQuery 表达式的 **where** 子句中，将 SQL 查询生成的文档进一步限制为 <city> 元素（沿指定的路径）的值为 Toronto 的那些文档。

此查询产生的结果与使用 db2-fn:xmlcolumn 的前一个示例产生的结果相同。

```
<out xmlns="http://posample.org">
<name>
              Kathy Smith
</name>
</out>
```

使用 **db2-fn:sqlquery** 时附带参数

要将值传递至 db2-fn:sqlquery 函数中的 SQL 全查询，请运行以下查询:

```
XQUERY declare default element namespace "http://posample.org";
  let $testval := 1000
  for $d in db2-fn:sqlquery(
    'SELECT INFO FROM CUSTOMER WHERE Cid = parameter(1)',
    $testval)/customerinfo
  return <out>{$d/name}</out>~
```

在 XQuery 表达式中，**let** 子句将 `$testval` 的值设置为 1000。然后，在 **for** 子句中，表达式会使用 PARAMETER 标量函数将该值传递至 db2-fn:sqlquery 函数。

XQuery 表达式将返回以下已构造元素:

```
<out xmlns="http://posample.org">
   <name>Kathy Smith</name>
</out>
```

返回至教程

课程 7: 针对 XML 模式验证 XML 文档

本课程描述如何验证 XML 文档。只能针对 XML 模式验证 XML 文档；不支持针对 DTD 进行验证。虽然您不能针对 DTD 进行验证，但是仍然可以插入包含 DOCTYPE 或者引用 DTD 的文档。

可以使用一些工具（例如，IBM Rational® Application Developer 中的工具）来帮助您根据各种源（其中包括 DTD、表和 XML 文档）生成 XML 模式。

在验证之前，必须向内置 XML 模式存储库（XSR）注册 XML 模式。此过程涉及到注册组成 XML 模式的每个 XML 模式文档，然后完成注册。一种注册 XML 模式的方法是使用命令。

要注册模式文档和完成 posample.customer XML 模式的注册，请运行以下命令。（因为此 XML 模式仅由一个模式文档组成，所以可使用单个命令来注册文档和完成注册。）此命令将指定 sqllib/samples/xml 目录的绝对路径。如果系统上的路径不是以 c:/sqllib/ 开头，那么请在命令中相应地修改文件路径。

```
REGISTER XMLSCHEMA 'http://posample.org'
FROM 'file:///c:/sqllib/samples/xml/customer.xsd' AS posample.customer COMPLETE~
```

通过查询 SYSCAT.XSROBJECTS 目录视图（它包含有关存储在 XSR 中的对象的信息），就可以验证是否成功注册了该 XML 模式。为了更加清楚明了，对此查询及其结果进行了编排，如下所示:

```
SELECT OBJECTSCHEMA, OBJECTNAME FROM SYSCAT.XSROBJECTS~


OBJECTSCHEMA          OBJECTNAME
--------------------  --------------------
POSAMPLE              CUSTOMER
```

现在，可以使用 XML 模式来进行验证。通常，在执行 INSERT 或 UPDATE 操作期间使用 XMLVALIDATE 函数来执行验证。仅当验证成功之后，才会执行对其指定了 XMLVALIDATE 的 INSERT 或 UPDATE 操作。

要将 XML 文档插入 CUSTOMER 表的 INFO 列（要求此文档对于 posample.customer XML 模式有效），请发出以下语句:

```
INSERT INTO Customer(Cid, Info) VALUES (1003, XMLVALIDATE (XMLPARSE (DOCUMENT
'<customerinfo xmlns="http://posample.org" Cid="1003">
  <name>Robert Shoemaker</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>' PRESERVE WHITESPACE )
ACCORDING TO XMLSCHEMA ID posample.customer ))~
```

此示例中的 XML 文档作为字符数据传递。但是，XMLVALIDATE 仅对 XML 数据进行操作。因为 XML 文档作为字符数据来传递，所以您必须使用 XMLPARSE 函数来显式解析数据。XMLPARSE 函数将其自变量解析为 XML 文档并返回 XML 值。

DB2 数据库服务器对某些操作执行隐式解析。例如，当您在 INSERT、UPDATE、DELETE 或 MERGE 语句中将数据类型为 STRING（字符、图形或二进制）的主变量、参数标记或 SQL 表达式指定给 XML 列时，就会执行隐式解析。。

要验证是否成功完成了插入和验证操作，请查询 INFO 列:

```
SELECT Info FROM Customer~
```

此查询应返回三个 XML 文档，其中一个是您刚插入的文档。

返回至教程

课程 8: 使用 XSLT 样式表进行变换

本课程描述如何使用可扩展样式表语言变换（XSLT）样式表和内置函数 XSLTRANSFORM 将数据库中的 XML 数据转换为其他格式。

以一个包含任意数目的大学生记录的 XML 文档为例。每个 student 元素包含学生的标识、名字、姓氏、年龄以及就读的大学。以下文档包含两个学生:

```
<?xml version="1.0"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <student studentID="1" givenName="Steffen" familyName="Siegmund"
        age="21" university="Rostock"/>
     <student studentID="2" givenName="Helena" familyName="Schmidt"
         age="23" university="Rostock"/>
</students>
```

此外，假定您希望抽取 XML 记录中的信息并创建可在浏览器中查看的 HTML Web 页面。要变换信息，需要下列 XSLT 样式表:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="showUniversity"/>
<xsl:template match="students">
     <html>
     <head/>
     <body>
     <h1><xsl:value-of select="$headline"/></h1>
     <table border="1">
     <th>
     <tr>
     <td width="80">StudentID</td>
     <td width="200">Given Name</td>
     <td width="200">Family Name</td>
     <td width="50">Age</td>
     <xsl:choose>
          <xsl:when test="$showUniversity =''true''">
               <td width="200">University</td>
          </xsl:when>
     </xsl:choose>
     </tr>
     </th>
     <xsl:apply-templates/>
     </table>
     </body>
     </html>
     </xsl:template>
            <xsl:template match="student">
               <tr>
               <td><xsl:value-of select="@studentID"/></td>
               <td><xsl:value-of select="@givenName"/></td>
               <td><xsl:value-of select="@familyName"/></td>
               <td><xsl:value-of select="@age"/></td>
               <xsl:choose>
                    <xsl:when test="$showUniversity = ''true'' ">
                         <td><xsl:value-of select="@university"/></td>
                    </xsl:when>
```

```
            </xsl:choose>
            </tr>
     </xsl:template>
</xsl:stylesheet>
```

要变换数据:

1. 通过运行下列命令来创建两个用于存储 XML 文档和样式表文档的表:

   ```
   CREATE TABLE XML_DATA (DOCID INTEGER, XML_DOC XML )~
   CREATE TABLE XML_TRANS (XSLID INTEGER, XSLT_DOC CLOB(1M))~
   ```

2. 使用下列 INSERT 语句将 XML 文档和整个 XSLT 样式表插入表中。

   为了简洁明了起见，在此步骤中，第二个 INSERT 语句中显示了已截断的 XSLT 样式表。在使用此语句之前，请将已截断的样式表替换为先前列示的 XSLT 样式表。

   ```
   INSERT INTO XML_DATA VALUES
        (1,
         '<?xml version="1.0"?>
          <students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
             <student studentID="1" givenName="Steffen" familyName="Siegmund"
               age="21" university="Rostock"/>
             <student studentID="2" givenName="Helena" familyName="Schmidt"
            age="23" university="Rostock"/>
         </students>'
    )~

   INSERT INTO XML_TRANS VALUES
        (1,
        '<?xml version="1.0" encoding="UTF-8"?>
        <xsl:stylesheet version="1.0"
             ...
        </xsl:stylesheet>'
   )~
   ```

3. 通过调用 XSLTRANSFORM 函数来变换 XML 文档:

   ```
   SELECT XSLTRANSFORM (XML_DOC USING XSLT_DOC AS CLOB(1M))
      FROM XML_DATA, XML_TRANS WHERE DOCID = 1 and XSLID = 1 ~
   ```

此变换的输出为以下 HTML 文件:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">Given Name</td>
<td width="200">Family Name</td>
<td width="50">Age</td>
</tr>
</th>
<tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>21</td>
</tr>
<tr>
<td>2</td><td>Helena</td><td>Schmidt</td>
<td>23</td>
```

```
</tr>
</table>
</body>
</html>
```

您可能希望改变 XSLT 样式表在运行时的行为，以添加未包含在 XML 记录中的信息或者更改输出本身的性质（例如，更改为创建 XHTML 输出而不是标准 HTML 输出）。要改变行为，可以使用参数文件将参数传递至 XSLT 进程。参数文件本身是 XML 文档，并且包含对应于 XSLT 样式表文件中类似语句的 param 语句。

考虑在样式表中定义的、但是在前一变换中未使用的以下两个参数:

```
<xsl:param name="showUniversity"/>
<xsl:param name="headline"/>
```

为了使用这些参数来变换 XML 文档，可以将参数文件存储在表中并将此文件与 XSLTRANSFORM 函数配合使用。

1. 创建 PARAM_TAB 表来存储参数文件:

   ```
   CREATE TABLE PARAM_TAB (DOCID INTEGER, PARAM VARCHAR(1000))~
   ```

2. 按如下所示创建参数文件:

   ```
   INSERT INTO PARAM_TAB VALUES
        (1,
        '<?xml version="1.0"?>
        <params xmlns="http://www.ibm.com/XSLTransformParameters">
             <param name="showUniversity" value="true"/>
             <param name="headline">The student list</param>
        </params>'
        )~
   ```

3. 通过调用 XSLTRANSFORM 函数来变换 XML 文档:

   ```
   SELECT XSLTRANSFORM (XML_DOC USING XSLT_DOC WITH PARAM AS CLOB(1M) )
      FROM XML_DATA X , PARAM_TAB P, XML_TRANS
      WHERE X.DOCID=P.DOCID and XSLID = 1 ~
   ```

此过程的输出为以下 HTML 文件:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1>The student list</h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">Given Name</td>
<td width="200">Family Name</td>
<td width="50">Age</td>
<td width="200">University</td>
</tr>
</th>
<tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>21</td>
<td>Rostock</td>
</tr>
<tr>
<td>2</td>
<td>Helena</td><td>Schmidt</td>
<td>23</td>
```

```
<td>Rostock</td>
</tr>
</table>
</body>
</html>
```

返回至教程

第 3 章 XML 存储器

插入到类型为 XML 的列中的 XML 文档可驻留在缺省存储对象或直接存储在基本表行中。基本表行存储由您控制，并且仅可用于小型文档；较大的文档总是存储在缺省存储对象中。

是否将文档存储在基本表行中取决于存储和性能要求以及接受的方式。

XML 存储对象
: 这是存储 XML 文档的缺省方法。不管您选择如何存储，需要 32 KB 以上的存储空间或者超过了页大小的文档总是存储在缺省存储对象中。存储在缺省存储对象中允许您插入和检索最大 2 GB 的 XML 文档。

基本表行存储
: 对于所需存储空间少于 32 KB 的 XML 文档，可以选择直接将 XML 文档存储在基本表行中。因为需要的 I/O 操作减少，所以此选项会提高查询、插入、更新或删除 XML 文档的性能。

如果对此表启用数据行压缩，那么存储在缺省 XML 存储对象和基本表行中的 XML 文档将进行压缩。经过压缩之后，可以提高对 XML 文档执行 I/O 操作的效率，还可以减少需要的存储空间。

XML 存储对象

缺省情况下 DB2 数据库服务器将类型为 XML 的表列中包含的 XML 文档存储在 XML 存储对象中，LOB 数据以相似方式存储在不同于表的其他内容的位置。

XML 存储对象与它们的父表对象是分开的，但存储对象依赖于父表对象。对于存储在 XML 表列的行中的每个 XML 值，DB2 都会维护一条称为 XML 数据说明符（XDS）的记录，该记录指定从关联的 XML 存储对象中的何处检索存储在磁盘上的 XML 数据。存储在系统管理空间中时，与 XML 存储对象关联的文件具有文件类型扩展名 .xda。有时将 XML 存储对象称为 XML 数据区（XDA）。

最多可以将大小为 2 吉字节的 XML 文档存储在数据库中。因为 XML 数据可能非常大，所以可能要与其他数据的缓冲活动分开单独监视 XML 数据的缓冲活动。提供了一些监视元素来帮助您监视 XML 存储对象的缓冲池活动。

有关使用 XML 存储对象的 XML 列所需的空间的其他信息，请参阅 XML 列的“字节计数”（未在“CREATE TABLE 语句”中指定 INLINE LENGTH）。

XML 基本表行存储器

可选择将小型和中型 XML 文档存储在基本表行中，而不是将它们存储在缺省 XML 存储对象中。XML 文档的行存储类似于结构化类型实例以直接插入的方式存储在表行中的情况。

在启用基本表行存储之前，需要决定要将多少行空间用于每个 XML 列的行存储。可提供的空间取决于可用的最大行大小，而最大行大小又取决于创建表的表空间的页大小

以及指定为表的一部分的其他列。要计算可用的行空间，请参阅 XML 列的“行大小”和“字节计数”以及“CREATE TABLE 语句”中指定的“INLINE LENGTH”。

启用基本表行存储

可在创建包含 XML 列的表或改变包含 XML 列的现有表时，指定XML 文档应存储在基本表行中而不是存储在缺省 XML 存储对象中要启用基本表行存储，对于应使用行存储的每个 XML 列，需要将 INLINE LENGTH 关键字与 CREATE TABLE 或 ALTER TABLE 语句包括在一起，后跟要存储在基本表行中的 XML 文档的最大大小（以字节计）。

注意，改变现有表的 XML 列不会将已经存储在该列中的 XML 文档自动移到基本表行中。要移动 XML 文档，必须使用 UPDATE 语句更新所有 XML 文档。

限制

基本表行存储仅可供内部表示不超过 32 KB（如果行大小较小，那么会更小）的 XML 文档使用，同时需要减去指定了 INLINE LENGTH 选项的 XML 列的必需字节计数开销。32 KB 大小假定表空间页大小为 32 KB。存储超过指定直接插入长度的 XML 文档时，超过大小的文档将自动存储在缺省 XML 存储对象中。

一旦对 XML 列指定了直接插入长度，就只能提高用于 XML 文档的行存储的直接插入长度大小，但不能降低该大小。

示例

以下示例对 SAMPLE 数据库中的 PRODUCT 表的 XML 列 DESCRIPTION 启用 XML 文档的基本表行存储。此示例将要存储在基本表行中的 XML 文档的最大直接插入长度设置为 32000 字节，这会为开销留下额外所需空间。使用值“32000 字节”时假定表空间页大小为 32 KB。改变 XML 列后，UPDATE 语句会将 XML 文档移到基本表行中。

```
ALTER TABLE PRODUCT
   ALTER COLUMN DESCRIPTION
   SET INLINE LENGTH 32000

UPDATE PRODUCT SET DESCRIPTION = DESCRIPTION
```

以下示例创建的 MYCUSTOMER 表与 SAMPLE 数据库的 CUSTOMER 表类似，但其基本表行存储是对 XML 列 Info 指定的。插入到 INFO 列中时，内部表示不超过 2000 字节的文档将存储在基本表行中。

```
CREATE TABLE MYCUSTOMER (Cid BIGINT NOT NULL,
   Info XML INLINE LENGTH 2000,
   History XML,
   CONSTRAINT PK_CUSTOMER PRIMARY KEY (Cid)) in IBMDB2SAMPLEXML
```

XML 文档的存储要求

XML 文档在 DB2 数据库中所占用的空间大小由原始格式的文档的最初大小和一些其他因素确定。

以下列表包含这些因素中最重要的部分:

文档结构

包含复杂标记的 XML 文档需要的存储器空间比具有简单标记的文档所需的空

间要大。例如，如果一个 XML 文档具有许多嵌套元素，每个嵌套元素包含少量文本或具有简短的属性值，那么该文档占用的存储器空间比主要由文本内容组成的 XML 文档要多。

节点名 元素名称、属性名称、名称空间前缀以及类似的非内容数据的长度也影响存储器的大小。压缩原始格式超过 4 字节的任何这种类型的信息单元以进行存储，将使得存储器效率比使用较长的节点名要高。

属性数与元素数之比
通常，每个元素使用的属性越多，XML 文档所需的存储器空间大小就越小。

文档代码页
如果 XML 文档所使用的编码使得每个字符要使用多个字节，那么该 XML 文档占用的存储器空间大小比使用单字节字符集的文档要大。

压缩 如果对包含 XML 列的表启用数据行压缩，那么 XML 文档需要的存储空间更少。

如果表中包含使用 DB2 版本 9.5 或更低版本的 XML 记录格式的 XML 列，那么不支持压缩该表的 XML 存储对象中的数据。如果对这样的表启用数据行压缩，那么将只压缩表对象中的表行数据。要使表的 XML 存储对象中的数据可供压缩，请使用 ADMIN_MOVE_TABLE 存储过程来迁移此表，然后启用数据行压缩。

第 4 章 插入 XML 数据

必须先创建包含 XML 列的表，或向现有表添加 XML 列，才能插入 XML 文档。

创建具有 XML 列的表

要创建具有 XML 列的表，在 CREATE TABLE 语句中指定 XML 数据类型的列。一个表可以有一个或多个 XML 列。

定义 XML 列时不要指定长度。但是，与 DB2 数据库交换的已序列化 XML 数据的大小限制为每个 XML 类型的值有 2 GB，因此，XML 文档的有效限制为 2 GB。

与 LOB 列一样，XML 列仅包含列的描述符。数据单独存储。

注:

- 如果对表启用数据行压缩，那么 XML 文档需要的存储空间更少。
- 可选择将小型和中型 XML 文档存储在基本表行中，而不是将它们存储在缺省 XML 存储对象中。

示例: 样本数据库包含一个客户数据表，它包含两个 XML 列。定义如下所示:

```
CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY,
                       Info XML,
                       History XML)
```

示例: VALIDATED 谓词检查是否验证了指定的 XML 列中的值。可以使用 VALIDATED 谓词定义对 XML 列的表检查约束，以确保在表中插入或更新的所有文档都有效。

```
CREATE TABLE TableValid (id BIGINT,
                         xmlcol XML,
                         CONSTRAINT valid_check CHECK (xmlcol IS VALIDATED))
```

示例: 将 COMPRESS 属性设置为 YES 就会启用数据行压缩。存储在 XML 列中的 XML 文档将进行行压缩。压缩行级别的数据时，允许将重复的模式替换为更短的符号字符串。

```
CREATE TABLE TableXmlCol (id BIGINT,
                         xmlcol XML) COMPRESS YES
```

示例: 以下 CREATE TABLE 语句创建按探访日期分区的患者表。介于 2000 年 1 月 1 日到 2006 年 12 月 31 日之间的所有记录在第一个分区内。最新的数据按每 6 个月来创建分区。

```
CREATE TABLE Patients ( patientID BIGINT, visit_date DATE, diagInfo XML,
    prescription XML )
    INDEX IN indexTbsp LONG IN ltbsp
    PARTITION BY ( visit_date )
       ( STARTING '1/1/2000' ENDING '12/31/2006',
         STARTING '1/1/2007' ENDING '6/30/2007',
            ENDING '12/31/2007',
            ENDING '6/30/2008',
            ENDING '12/31/2008',
            ENDING '6/30/2009' );
```

将 XML 列添加至现有表

要将 XML 列添加到现有表，可在带有 ADD 子句的 ALTER TABLE 语句中指定数据类型为 XML 的列。一个表可以有一个或多个 XML 列。

示例 样本数据库包含一个客户数据表，它包含两个 XML 列。定义如下所示:

```
CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY,
                                              Info XML,
                                              History XML)
```

创建一个名为 MyCustomer 的表作为 Customer 的副本，并添加一个 XML 列来描述客户的喜好。

```
CREATE TABLE MyCustomer LIKE Customer;
ALTER TABLE MyCustomer ADD COLUMN Preferences XML;
```

示例: 将 COMPRESS 属性设置为 YES 就会启用数据行压缩。存储在 XML 列中的 XML 文档将进行行压缩。压缩行级别的数据时，允许将重复的模式替换为更短的符号字符串。

```
ALTER TABLE MyCustomer ADD COLUMN Preferences XML COMPRESS YES;
```

示例: 以下 CREATE TABLE 语句创建按探访日期分区的患者表。介于 2000 年 1 月 1 日到 2006 年 12 月 31 日之间的所有记录在第一个分区内。最新的数据按每 6 个月来创建分区。

```
CREATE TABLE Patients ( patientID INT, Name Varchar(20), visit_date DATE,
      diagInfo XML )
    PARTITION BY ( visit_date )
       ( STARTING '1/1/2000' ENDING '12/31/2006',
         STARTING '1/1/2007' ENDING '6/30/2007',
            ENDING '12/31/2007',
            ENDING '6/30/2008',
            ENDING '12/31/2008',
            ENDING '6/30/2009' );
```

以下 ALTER 表语句添加有关患者处方信息的另一 XML 列:

```
ALTER TABLE Patients ADD COLUMN prescription XML ;
```

插入到 XML 列中

要将数据插入到 XML 列中，可使用 SQL INSERT 语句。XML 列的输入必须是格式良好的 XML 文档，如 XML 1.0 规范中所定义的那样。应用程序数据类型可以是 XML、字符或二进制类型。

建议通过主变量而不是字面值插入 XML 数据，以便 DB2 数据库服务器可以使用主变量数据类型来确定一些编码信息。

应用程序中的 XML 数据采用其序列化字符串格式。将数据插入到 XML 列中时，它必须转换为其 XML 分层格式。如果应用程序数据类型是 XML 数据类型，那么 DB2 数据库服务器将隐式执行此操作。如果应用程序数据类型不是 XML 类型，那么在执行插入操作时可显式调用 XMLPARSE 函数，以将数据从其序列化字符串格式转换为 XML 分层格式。

在文档插入期间，还可能想要针对已注册的 XML 模式验证 XML 文档。可以使用 XMLVALIDATE 函数来执行此操作。

下列示例说明了如何将 XML 数据插入到 XML 列中。这些示例使用表 MyCustomer，它是样本 Customer 表的副本。要插入的 XML 数据在文件 c6.xml 中，并且看起来如下所示:

```
<customerinfo Cid="1015">
  <name>Christine Haas</name>
  <addr country="Canada">
    <street>12 Topgrove</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X-7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-5238</phone>
  <phone type="home">416-555-2934</phone>
</customerinfo>
```

示例: 在 JDBC 应用程序中，以二进制数据的形式读取文件 c6.xml 中的 XML 数据，并将数据插入到 XML 列中:

```
PreparedStatement insertStmt = null;
String sqls = null;
int cid = 1015;
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
File file = new File("c6.xml");
insertStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
insertStmt.executeUpdate();
```

示例: 在静态嵌入式 C 应用程序中，将数据从二进制 XML 主变量插入到 XML 列中:

```
EXEC SQL BEGIN DECLARE SECTION;
   sqlint64 cid;
   SQL TYPE IS XML AS BLOB (10K) xml_hostvar;
EXEC SQL END DECLARE SECTION;
...
cid=1015;
/* Read data from file c6.xml into xml_hostvar */
...
EXEC SQL INSERT INTO MyCustomer (Cid,Info) VALUES (:cid, :xml_hostvar);
```

XML 解析

XML 解析是将 XML 数据从其序列化字符串格式转换为分层格式的过程。

可以让 DB2 数据库服务器隐式执行解析，也可以显式执行 XML 解析。

在下列情况下进行*隐式 XML 解析*:

- 使用类型为 XML 的主变量或使用类型为 XML 的参数标记将数据传递至数据库服务器时

  数据库服务器在绑定主变量或参数标记的值以便在语句处理期间使用时进行解析

  在这种情况下，必须使用隐式解析。
- 在 INSERT、UPDATE、DELETE 或 MERGE 语句中将字符串数据类型（character、graphic 或 binary）的主变量、参数标记或 SQL 表达式指定给 XML 列时。当 SQL 编译器隐式将 XMLPARSE 函数添加至该语句时进行解析。

对输入 XML 数据调用 XMLPARSE 函数时，执行*显式 XML 解析*。可以在接受 XML 数据类型的任何上下文中使用 XMLPARSE 的结果。例如，可以将结果指定给 XML 列或将它用作类型为 XML 的存储过程参数。

XMLPARSE 函数采用非 XML、字符或二进制数据类型作为输入。对于嵌入式动态 SQL 应用程序，需要将表示 XMLPARSE 的输入文档的参数标记转换为相应的数据类型。例如:

```
INSERT INTO MyCustomer (Cid, Info)
  VALUES (?, xmlparse(document cast(? as clob(1k)) preserve whitespace))
```

对于静态嵌入式 SQL 应用程序，不能将 XMLPARSE 函数的主变量自变量声明为 XML 类型（XML AS BLOB、XML AS CLOB 或 XML AS DBCLOB 类型）。

XML 解析和空格处理

在隐式或显式 XML 解析期间，将数据存储在数据库中时，可以控制是保留还是去掉边界空格字符。

根据 XML 标准，空格是文档中用于提高可读性的间隔字符（U+0020）、回车符（U+000D）、换行符（U+000A）或制表符（U+0009）。当任何这些字符作为文本字符串的一部分出现时，不将它们视为空格。

*边界空格*是出现在元素之间的空格字符。例如，在以下文档中，<a> 与 以及 与 之间的空格是边界空格。

```
<a> <b> and between </b> </a>
```

通过显式调用 XMLPARSE，可以使用 STRIP WHITESPACE 或 PRESERVE WHITESPACE 选项来控制是否保留边界空格。缺省行为是去掉边界空格。

通过隐式 XML 解析:

- 如果输入数据类型不是 XML 类型或未转换为 XML 数据类型，那么 DB2 数据库服务器总是去掉空格。
- 如果输入数据类型是 XML 数据类型，那么可以使用 CURRENT IMPLICIT XMLPARSE OPTION 专用寄存器来控制是否保留边界空格。可以将此专用寄存器设置为 STRIP WHITESPACE 或 PRESERVE WHITESPACE。缺省行为是去掉边界空格。

如果使用 XML 验证，那么 DB2 数据库服务器将忽略 CURRENT IMPLICIT XMLPARSE OPTION 专用寄存器，并只使用验证规则来确定下列示例中是去掉还是保留空格:

```
xmlvalidate(? ACCORDING TO XMLSCHEMA ID schemaname)
xmlvalidate(?)
xmlvalidate(:hvxml ACCORDING TO XMLSCHEMA ID schemaname)
xmlvalidate(:hvxml)
xmlvalidate(cast(? as xml) ACCORDING TO XMLSCHEMA ID schemaname)
xmlvalidate(cast(? as xml))
```

在此情况下，? 表示 XML 数据，而 :hvxml 是 XML 主变量。

有关 XML 验证如何影响空格处理方式的信息，请参阅 XML 验证。

XML 标准指定 xml:space 属性，它用于控制是去掉还是保留 XML 数据中的空格。xml:space 属性覆盖任何空格设置以进行隐式或显式 XML 解析。

例如，在以下文档中，无论 XML 解析选项如何，总是保留正好在 <b> 前后的空格，因为这些空格位于具有属性 xml:space="preserve" 的节点内：

```
<a xml:space="preserve"> <b> <c>c</c>b </b></a>
```

但是，在以下文档中，可以用 XML 解析选项控制正好在 <b> 前后的空格，因为这些空格位于具有属性 xml:space="default" 的节点内：

```
<a xml:space="default"> <b> <c>c</c>b </b></a>
```

非 Unicode 数据库中的 XML 解析

将 XML 文档传递至非 Unicode 数据库时，首先文档从客户机传递至目标数据库服务器时会进行代码页转换，然后在文档传递至 DB2 XML 解析器时会进行代码页转换。使用类型为 XML 的主变量或参数标记传递 XML 文档可避免进行代码页转换。如果使用字符数据类型（CHAR、VARCHAR、CLOB 或 LONG VARCHAR）传递 XML 文档，那么代码页转换可能导致引入 XML 数据内未包含在目标数据库代码页中的任何字符的替换字符。

为避免引入替换字符及已插入 XML 数据的潜在降低，应确保使用字符数据类型解析 XML 数据时，源文档中的所有代码点都在目标数据库代码页中。对于不在此代码页中的任何字符，可使用十进制或十六进制字符实体引用来指定正确的 Unicode 代码点。例如，&#x003E or &#0062 可用来指定 >（大于号）字符。

还可使用 ENABLE_XMLCHAR 配置参数来控制是否对字符数据类型启用 XML 解析。将 ENABLE_XMLCHAR 设置为“NO”可阻止使用字符数据类型时的显式和隐式 XML 解析。

XML 解析和 DTD

如果输入数据包含内部文档类型声明（DTD）或引用外部 DTD，那么 XML 解析过程还会检查这些 DTD 的语法。此外，解析过程还：
- 应用内部和外部 DTD 定义的缺省值
- 扩展实体引用和参数实体

示例

以下示例说明在不同情况下如何处理 XML 文档中的空格。

示例：文件 c8.xml 包含以下文档：

```
<customerinfo xml:space="preserve" Cid='1008'>
  <name>Kathy Smith</name>
  <addr country='Canada'>
    <street>14 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-3333</phone>
</customerinfo>
```

在 JDBC 应用程序中，从文件中读取 XML 文档，然后将数据插入到表 MYCUSTOMER 的 XML 列 INFO 中，该表是样本 Customer 表的副本。让 DB2 数据库服务器执行隐式 XML 解析操作。

```
PreparedStatement insertStmt = null;
String sqls = null;
int cid = 1008;
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
File file = new File("c8.xml");
insertStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
insertStmt.executeUpdate();
```

未指定空格处理方式，因此采用缺省行为：去掉空格。但是，文档包含 xml:space="preserve" 属性，因此保留空格。这表示将保留文档中元素之间的回车符、换行符和空格。

如果检索存储的数据，那么内容看起来如下所示：

```
<customerinfo xml:space="preserve" Cid='1008'>
  <name>Kathy Smith</name>
  <addr country='Canada'>
    <street>14 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-3333</phone>
</customerinfo>
```

示例：假定以下文档位于 BLOB 主变量 blob_hostvar 中。

```
<customerinfo xml:space="default" Cid='1009'>
  <name>Kathy Smith</name>
  <addr country='Canada'>
    <street>15 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-4444</phone>
</customerinfo>
```

在静态嵌入式 C 应用程序中，将主变量中的文档插入到表 MyCustomer 的 XML 列 Info 中。该主变量不是 XML 类型，因此需要显式执行 XMLPARSE。指定 STRIP WHITESPACE 以除去任何边界空格。

```
EXEC SQL BEGIN DECLARE SECTION;
   SQL TYPE BLOB (10K) blob_hostvar;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL INSERT INTO MyCustomer (Cid, Info)
  VALUES (1009,
  XMLPARSE(DOCUMENT :blob_hostvar STRIP WHITESPACE));
```

文档包含 xml:space="default" 属性，因此指定了 STRIP WHITESPACE 的 XMLPARSE 将控制空格处理方式。这表示将除去文档中元素之间的回车符、换行符和空格。

如果检索存储的数据，那么您将看到具有以下内容的单个行：

```
<customerinfo xml:space="default" Cid='1009'>
<name>Kathy Smith</name><addr country='Canada'><street>15 Rosewood</street>
<city>Toronto</city><prov-state>Ontario</prov-state><pcode-zip>M6W 1E6</pcode-zip>
</addr><phone type='work'>416-555-4444</phone></customerinfo>
```

示例: 在 C 语言应用程序中，主变量 clob_hostvar 包含以下文档，该文档包含内部 DTD:

```
<!DOCTYPE prod [<!ELEMENT description (name,details,price,weight)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT details (#PCDATA)>
   <!ELEMENT price (#PCDATA)>
   <!ELEMENT weight (#PCDATA)>
   <!ENTITY desc  "Anvil">
]>
<product pid=''110-100-01'' >
 <description>
 <name>&desc;</name>
 <details>Very heavy</details>
 <price>        9.99           </price>
 <weight>1 kg</weight>
 </description>
</product>'
```

将数据插入到表 MYPRODUCT 中，该表是样本 PRODUCT 表的副本:

```
EXEC SQL BEGIN DECLARE SECTION;
   SQL TYPE CLOB (10K) clob_hostvar;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL insert into
  Product ( pid, name, Price, PromoPrice, PromoStart, PromoEnd, description )
  values ( '110-100-01','Anvil', 9.99, 7.99, '11-02-2004','12-02-2004',
  XMLPARSE ( DOCUMENT :clob_hostvar STRIP WHITESPACE ));
```

XMLPARSE 指定去掉空格，因此将除去文档内的边界空格。此外，在数据库服务器执行 XMLPARSE 时，它将实体引用 `&desc;` 替换为它的值。

如果检索存储的数据，那么您将看到具有以下内容的单个行:

```
<product pid="110-100-01"><description><name>Anvil
</name><details>Very heavy</details><price>        9.99            </price>
<weight>1 kg</weight></description></product>
```

XML 数据完整性

需要确保 XML 文档遵循特定规则或符合特定处理要求，可执行附加 XML 数据完整性检查或指定执行操作前必须符合的附加条件。提供了确保 XML 数据完整性的一些不同方法，所选的方法取决于特定数据完整性和处理要求。

如果要为 XML 文档建立索引，还可强制在所有文档的 XML 列中唯一，其节点由您对其建立索引的 XML 模式限定。请参阅“UNIQUE 关键字语义”以了解更多信息。

XML 列的检查约束

检查约束允许您对 XML 列设置特定约束。每次尝试在 XML 列中插入或更新数据时会强制实施此约束；仅当符合约束指定的条件时才会执行此操作。

处理 XML 文档时，一个重要注意事项是先前是否对 XML 模式验证了这些文档。如果需要确保仅查询、插入、更新或删除符合特定验证条件的文档，那么使用 VALIDATED 谓词来提供条件。注意，检查约束决不会验证 XML 文档，它只会测试是否验证了 XML 文档。[2]

VALIDATED 谓词检查 *XML-expression* 指定的值的验证状态，该值必须为 XML 数据类型。如果未指定可选 according-to-clause，那么用于验证的 XML 模式不会影响结果。检查约束不会验证 XML 文档本身；约束仅测试文档的当前验证状态（IS VALIDATED 或 IS NOT VALIDATED）。如果指定了 according-to-clause，那么用于验证 *XML-expression* 指定的值的 XML 模式必须是 according-to-clause 标识的 XML 模式。您需要先向 XML 模式存储库注册 XML 模式，才能在 VALIDATED 谓词中引用这些模式。

注意:

- 检查约束依赖于它们引用的 XML 模式。如果已删除 XML 模式的 XSR 对象，那么还会删除引用该模式的任何约束。
- XML 列支持 NOT NULL 约束。
- XML 列支持对 XML 验证定义的信息约束。

检查约束求值

检查约束根据 IS VALIDATED 谓词的输出测试文档的验证状态。如果满足指定的条件，那么约束求值为 true，如果不满足指定的条件，那么输出求值为 false。如果 *XML-expression* 指定的值为空，那么谓词结果未知。

如果 *XML-expression* 指定的值不为空并且出现以下情况，那么 VALIDATED 谓词的结果为 **true**:

- 未指定 according-to-clause 并且已验证 *XML-expression* 指定的值，或者
- 已指定 according-to-clause，并且已使用由 according-to-clause 标识的 XML 模式之一验证了 *XML-expression* 指定的值。

如果 *XML-expression* 指定的值不为空并且出现以下情况，那么该谓词的结果为 **false**:

- 未指定 according-to-clause 并且未验证 *XML-expression* 指定的值，或者
- 已指定 according-to-clause 并且未使用由 according-to-clause 标识的 XML 模式之一验证 *XML-expression* 指定的值。

在指定了可选 according-to-clause 的情况下，如果尚未验证 *XML-expression* 指定的值或者验证了 *XML-expression* 指定的值但未依据任何指定 XML 模式，那么 IS NOT VALIDATED 将返回 true。

等价表达式

VALIDATED 谓词

```
value1 IS NOT VALIDATED optional-clause
```

等价于搜索条件

```
NOT(value1 IS VALIDATED optional-clause)
```

2. 如果需要在 XML 列中存储 XML 文档之前自动验证 XML 列，可使用 BEFORE 触发器。

示例

示例: 仅选择已验证 XML 文档。假定在表 T1 中定义了列 XMLCOL。仅检索已经过任何 XML 模式验证的 XML 值。

```
SELECT XMLCOL FROM T1
  WHERE XMLCOL IS VALIDATED
```

示例: 强制实施以下规则: 除非经过验证, 否则不会插入或更新任何值。假定在表 T1 中定义了列 XMLCOL 并且将检查约束添加至 XMLCOL。

```
ALTER TABLE T1 ADD CONSTRAINT CK_VALIDATED
  CHECK (XMLCOL IS VALIDATED)
```

示例: 约束 INFO_CONSTRAINT 是信息性约束。不会强制执行以下规则: 除非经过验证, 否则不会插入或更新值。

```
CREATE TABLE xmltab (ID INT,
     DOC XML, CONSTRAINT INFO_CONSTRAINT CHECK (DOC IS VALIDATED) NOT ENFORCED)
```

信息性约束用于改进查询性能。

XML 数据的触发器处理

触发器对 INSERT、UPDATE 或 DELETE 语句执行的操作做出响应。处理 XML 数据时, 可使用 CREATE TRIGGER 语句对 XML 列创建带有 UPDATE 选项的 BEFORE 或 AFTER 触发器。还可对包括 XML 列的表创建带有 INSERT 或 DELETE 选项的 BEFORE 或 AFTER 触发器。

在触发器主体中, 引用受影响行中类型为 XML 的列的转换变量只能与 XMLVALIDATE 函数配合使用进行验证, 以将 XML 列值设置为 NULL, 或保留 XML 列值不变。

在 XML 列中存储 XML 文档之前, 可使用与 INSERT 或 UPDATE 语句配合使用的 BEFORE 触发器来自动验证这些文档。对注册 XML 模式验证 XML 文档是可选操作, 但强烈建议数据完整性不确定时这样做, 原因是这样可以确保只插入或更新有效的 XML 文档。

满足设置条件时会激活触发器; 如果未指定任何条件, 触发器总是处于激活状态。如果希望仅当必要时才触发对 XML 文档的验证, 那么可指定 XML 列的条件及 BEFORE 触发器的 WHEN 子句。在 WHEN 子句中, 可包括对 XML 文档的必需验证状态: 即这些文档必须已验证或者一定不能验证这些文档以激活触发器 (IS VALIDATED 或 IS NOT VALIDATED)。可选择通过指定 ACCORDING TO XMLSCHEMA 子句来包括一个或多个 XML 模式, 以告诉触发器在对约束求值时应考虑哪些 XML 模式。

注: 指定 WHEN 子句的触发器会导致额外开销。如果始终应在插入 XML 文档之前进行验证, 那么可省略 WHEN 子句。

引用 XML 模式的任何触发器依赖于该模式。必须先在 XML 模式存储库中注册 XML 模式, 然后才能引用 XML 模式。如果触发器所依赖的 XML 模式取后来被 XML 模式存储库删除, 那么触发器会被标记为不可操作。

示例: 创建以下 BEFORE 触发器: 在 SAMPLE 数据库的 PRODUCT 表中插入包含新产品描述的 XML 文档之前, 该触发器会自动验证这些文档。此触发器会在更新 XML 文档之前的任意时间激活。

```
CREATE TRIGGER NEWPROD NO CASCADE BEFORE INSERT ON PRODUCT
  REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
     SET (N.DESCRIPTION) = XMLVALIDATE(N.DESCRIPTION
     ACCORDING TO XMLSCHEMA URI 'http://posample.org/product.xsd');
  END
```

示例: 演进 XML 模式 product2.xsd 之后，已经存储的 XML 文档在演进模式下保证有效，只要它们对原始 XML 模式 product.xsd 有效。但是，您可能想要确保对这些 XML 文档的所有更新在演进模式 product2.xsd 下同样有效。向 XML 模式存储库注册 product2.xsd 以后，BEFORE UPDATE 触发器会在进行任何更新前验证 XML 文档:

```
CREATE TRIGGER UPDPROD NO CASCADE BEFORE UPDATE ON PRODUCT
  REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    SET (N.DESCRIPTION) = XMLVALIDATE(N.DESCRIPTION
    ACCORDING TO XMLSCHEMA ID product2);
  END
```

示例: 您想要在另一个表中记录插入或更新客户记录。这要求您创建两个触发器，一个 AFTER INSERT 用于新插入的记录，一个 AFTER UPDATE 用于更新的记录。在下面的示例中，可根据表 MyCustomer 的 XML 列信息创建触发器，该表是样本 Customer 表的副本。

首先对 MyCustomer 表创建 AFTER INSERT 触发器:

```
CREATE TRIGGER INSAFTR
  AFTER INSERT ON MyCustomer
  REFERENCING NEW AS N
  FOR EACH ROW
  BEGIN ATOMIC
    INSERT INTO CustLog VALUES(N.CID, CURRENT TIMESTAMP, 'Insert');
  END
```

然后对 MyCustomer 表创建 AFTER UPDATE 触发器:

```
CREATE TRIGGER UPDAFTR
  AFTER UPDATE OF Info
  ON MyCustomer
  REFERENCING NEW AS N
  FOR EACH ROW
  BEGIN ATOMIC
    INSERT INTO CustLog VALUES(N.CID, CURRENT TIMESTAMP, 'Update');
  END
```

XML 验证

XML 验证是确定 XML 文档的结构、内容和数据类型是否有效的过程。XML 验证也会去掉 XML 文档中的*可忽略空格*。验证操作是可选的，但强烈建议在数据完整性不确定时这样做，原因是这样可以确保 XML 文档在格式良好的同时遵守其 XML 模式提供的规则。

注意，只能针对 XML 模式验证 XML 文档。不能对 DTD 验证 XML 文档。

要验证 XML 文档，请使用 XMLVALIDATE 函数。可使用 SQL 语句指定 XMLVALIDATE 以在 DB2 数据库中插入或更新 XML 文档。为了自动验证 XML 文档，针对 XML 列的 BEFORE 触发器还可调用 XMLVALIDATE 函数。要强制验证 XML 文档，可创建检查约束。

必须先在内置 XML 模式存储库中注册构成 XML 模式的所有模式文档，然后才能调用 XMLVALIDATE 函数。XML 文档本身不必在数据库中就能够在 XMLVALIDATE 中进行验证:

XML 验证和可忽略空格

根据 XML 标准，空格是文档中用于提高可读性的间隔字符（U+0020）、回车符（U+000D）、换行符（U+000A）或制表符（U+0009）。当任何这些字符作为文本字符串的一部分出现时，不将它们视为空格。

*可忽略空格*是可从 XML 文档中除去的空格。XML 模式文档确定哪个空格是可忽略空格。如果 XML 文档定义仅元素复杂类型（仅包含其他元素的元素），那么可忽略元素之间的空格。如果 XML 模式定义包含非字符串类型的简单元素，那么可忽略该元素内的空格。

示例: 按如下所示定义样本 XML 模式文档 product.xsd 中的 `description` 元素:

```
<xs:element name="description" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="0" />
      <xs:element name="details" type="xs:string" minOccurs="0" />
      <xs:element name="price" type="xs:decimal" minOccurs="0" />
      <xs:element name="weight" type="xs:string" minOccurs="0" />
 ...
 </xs:complexType>
</xs:element>
```

`description` 元素具有仅元素复杂类型，因为它只包含其他元素。因此，`description` 元素中的元素之间的空格是可忽略空格。`price` 元素也可以包含可忽略空格，因为它是包含非字符串类型的简单元素。

在 XMLVALIDATE 函数中，可以显式指定要用于验证的 XML 模式文档。如果不指定 XML 模式文档，那么 DB2 数据库服务器将在输入文档中查找标识 XML 模式文档的 xsi:schemaLocation 或 xsi:noNamespaceSchemaLocation 属性。xsi:schemaLocation 或 xsi:noNamespaceSchemaLocation 属性由 XML 模式规范定义，称为 *XML 模式提示*。xsi:schemaLocation 属性包含一个或多个值对，它们可帮助查找 XML 模式文档。每个值对中的第一个值是名称空间，第二个值是提示，它指示名称空间的 XML 模式的位置。xsi:noNamespaceSchemaLocation 值只包含提示。如果 XML 模式文档是在 XMLVALIDATE 函数中指定的，那么它将覆盖 xsi:schemaLocation 或 xsi:noNamespaceSchemaLocation 属性。

下列示例假定 `product` 模式是在 XML 模式存储库（XSR）中注册的。可以使用诸如以下的 CLP 语句来完成注册:

```
REGISTER XMLSCHEMA http://posample.org/product.xsd FROM product.xsd \
AS myschema.product
COMPLETE XMLSCHEMA myschema.product
```

或者，因为 XML 模式由单个模式文档组成，所以可以使用单个语句来注册 XML 模式并完成注册:

```
REGISTER XMLSCHEMA http://posample.org/product.xsd FROM product.xsd \
AS myschema.product COMPLETE
```

示例: 假定按如下所示创建表 MyProduct:

```
CREATE TABLE MyProduct LIKE Product
```

您要使用动态 SQL 应用程序将以下文档插入到 MyProduct 表的 XML 列 Info 中，并且要针对 XML 模式文档 product.xsd 验证 XML 数据，该文档位于 MyProduct 表所在的数据库服务器上的 XML 模式存储库中。

```
<product xmlns="http://posample.org" pid=''110-100-01'' >
 <description>
 <name>Anvil</name>
 <details>Very heavy</details>
 <price>         9.99            </price>
 <weight>1 kg</weight>
 </description>
</product>'
```

在 INSERT 语句中，XMLVALIDATE 函数指定要用于验证的 XML 模式:

```
Insert into MyProduct
  (pid, name, Price, PromoPrice, PromoStart, PromoEnd, description)
  values ( '110-100-01','Anvil', 9.99, 7.99, '11-02-2004','12-02-2004',
 XMLVALIDATE(? ACCORDING TO XMLSCHEMA ID myschema.product))
```

检索存储的数据时，您可以看到 XMLVALIDATE 除去可忽略空格的位置。检索到的数据是具有以下内容的单个行:

```
<product xmlns="http://posample.org" pid="110-100-01"><description><name>Anvil
</name><details>Very heavy</details><price>9.99</price><weight>1 kg</weight>
</description></product>
```

`product` 模式定义 `name`、`details`、`price` 和 `weight` 元素周围的空格，其中 price 元素内的空格定义为可忽略空格，因此 XMLVALIDATE 将除去该空格。

如果需要确保仅将已验证的文档插入到 XML 列中，或仅从 XML 列中检索已验证的文档，那么使用 VALIDATED 谓词。

要测试是否在插入或更新 XML 文档前验证了该文档，请对 XML 列创建包含 VALIDATED 谓词的检查约束。要从 XML 列中仅检索已验证的文档，或仅检索已插入但未验证的文档，可在 WHERE 子句中使用 VALIDATED 谓词。如果需要检查是否根据特定 XML 模式验证了 XML 文档，请在 ACCORDING TO XMLSCHEMA 子句中使用 VALIDATED 谓词来包括 XML 模式。

VALIDATED 谓词还可用作触发器的一部分。要在 XML 列中插入或更新 XML 文档之前触发对尚未验证的 XML 文档的验证，请在 WHEN 子句中创建包含针对 XML 列的 VALIDATED 谓词的 BEFORE 触发器以调用 XMLVALIDATE 函数。

示例: 假定您想从 MyCustomer 表的 INFO 列中只检索已验证的 XML 文档。执行类似以下的 SELECT 语句:

```
SELECT Info FROM MyCustomer WHERE Info IS VALIDATED
```

示例: 假定您想将已验证的 XML 文档只插入到 MyCustomer 表的 INFO 列中。可以定义检查约束来强制执行此条件。按以下方式改变 MyCustomer 表:

```
ALTER TABLE MyCustomer ADD CONSTRAINT CK_VALIDATED CHECK (Info IS VALIDATED)
```

但是，发出此语句使得不必使用上一个示例中的 VALIDATED 谓词，因为只能在表中成功插入或更新有效文档。

示例: 假定您想要使用 customer 模式验证以下文档，但不想将它存储在数据库中。

```
<customerinfo xml:space="default"
  xmlns="http://posample.org"
  Cid='1011'>
  <name>Kathy Smith</name>
  <addr country='Canada'>
  <street>25 Rosewood</street>
  <city>Toronto</city>
  <prov-state>Ontario</prov-state>
  <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-6676</phone>
</customerinfo>
```

假定已将文档指定给应用程序变量。可以使用类似以下的 VALUES 语句来进行验证:

```
VALUES XMLVALIDATE(? according to xmlschema id myschema.customer)
```

根据 XML 模式此文档有效，因此 VALUES 语句返回包含该文档的结果表。如果该文档无效，那么 VALUES 返回 SQL 错误。

XSR_GET_PARSING_DIAGNOSTICS 存储过程

存储过程 XSR_GET_PARSING_DIAGNOSTICS 生成解析或验证 XML 文档时所发生错误的详细信息。此存储过程可用于报告解析错误和验证错误或仅报告解析错误。

如果在解析或验证 XML 文件期间发生错误，请从 DB2 命令窗口调用 XSR_GET_PARSING_DIAGNOSTICS 存储过程或者将此存储过程添加至应用程序。例如，使用 XMLVALIDATE 标量函数验证 XML 文档时，请使用此存储过程来生成验证期间所发生错误的详细信息。

语法

```
►►—XSR_GET_PARSING_DIAGNOSTICS—(—instance—,—rschema—,—name—,————————————►

►—schemaLocation—,—implicitValidation—,—errorDialog—,—errorCount—)————————►◄
```

此存储过程的模式为 SYSPROC。

权限

XML 模式授权: 用于验证的 XML 模式必须在 XML 模式存储库中注册之后才能使用。此存储过程的授权标识必须至少拥有下列其中一项特权和权限:

- 对用于验证的 XML 模式的 USAGE 特权
- DBADM 权限

过程参数

instance
: BLOB(30M) 类型的输入参数，它包含 XML 文档的内容。必须提供 XML 文档。此值不能为 NULL。

rschema
: VARCHAR(128) 类型的输入参数，它指定向 XML 模式存储库注册的两部分 XSR 对象名的 SQL 模式部分。此值可为 NULL。如果此值为 NULL，那么假定 SQL 模式部分是 CURRENT SCHEMA 专用寄存器的当前值。

name

VARCHAR(128) 类型的输入参数，它指定向 XML 模式存储库注册的两部分 XSR 对象名的模式名。XML 模式的完整 SQL 标识为 *rschema.name*。它对于 XSR 中的所有对象应该是唯一的。此值可为 NULL。

schemaLocation

VARCHAR(1000) 类型的输入参数，它指示 XML 模式主文档的模式位置。此参数是 XML 模式的外部名，即，可以在 XML 实例文档中使用 `xsi:schemaLocation` 属性来标识主文档。此值可为 NULL。

implicitValidation

INTEGER 类型的输入参数，它指示是否应该使用实例文档中的模式位置来查找 XML 模式。此值不能为 NULL。它的值可以为 0（表示"否"）和 1（表示"是"）。

0 不使用实例文档中的模式位置。如果传递的值为 0，那么可以使用下列其中一种方法来指定模式:

- 将 XSR 对象名作为在 XML 模式存储库中注册的模式的 *rschema* 和 *name* 参数来提供。
- 使用 *schemaLocation* 参数来提供模式位置。

如果同时指定了 XSR 对象名和 *schemaLocation*，那么将使用 XSR 对象名。如果未指定任何一项，那么不会执行验证。仅执行 XML 解析，并报告所有 XML 解析错误。

1 使用实例文档中的 `xsi:schemaLocation` 属性值的模式位置。

根据先前向 XML 模式存储库注册的 XML 模式文档来对输入文档执行验证。

如果 *implicitValidation* 参数的值为 0，而 *rschema*、*name* 和 *schemaLocation* 参数的值为 NULL，那么将解析实例文档但不会对它进行验证。

errorDialog

VARCHAR(32000) 类型的输出参数，它包含将列示解析和验证错误的 UTF-8 XML 文档。仅当至少有一个错误时，才会生成此文档。

errorCount

类型为 INTEGER 的输出参数，用于指定 XML 解析和验证错误的总数。

用法

可以通过以下三种方法来针对已注册的 XML 模式验证 XML 文档:

- 使用 *rschema* 和 *name* 参数为 XML 模式提供 XSR 对象名。
- 使用 *schemaLocation* 参数来提供模式位置。
- 如果 XML 实例文档将模式指定为 `xsi:schemaLocation` 属性的值，那么将 *implicitValidation* 设置为 1。

如果使用 XSR_GET_PARSING_DIAGNOSTICS 存储过程时发生解析或验证错误，那么设置 *errorDialog* 和 *errorCount* 输出参数。*errorDialog* 包含一个用于列示错误的 XML 文档。可以从使用 DB2 CLI、Java 或 C++ 的应用程序中调用 XSR_GET_PARSING_DIAGNOSTICS 存储过程，并使用参数标记来获取此存储过程的输出。

显示详细的 XML 解析和验证错误

可以使用 XSR_GET_PARSING_DIAGNOSTICS 存储过程中 **errorDialog** 输出参数中的信息来解决 XML 解析和验证错误。

如果使用 XMLVALIDATE 标量函数来验证 XML 文档时发生错误，那么请使用 XSR_GET_PARSING_DIAGNOSTICS 存储过程来生成详细的错误信息。以下示例将 XSR_GET_PARSING_DIAGNOSTICS 存储过程与简单的 XML 模式和 XML 文档配合使用来生成详细的验证错误信息。

样本 XML 模式

此示例使用以下 XML 模式定义（XSD）。此模式具有各种元素，其中 INTEGER 类型属性的元素指定给 Age 元素。

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://my.simpletest"
           xmlns="http://my.simpletest"
           elementFormDefault="qualified">
<xs:element name="Person">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="Name">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="FirstName" type="xs:string"/>
        <xs:element name="LastName" type="xs:string"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    <xs:element name="Age" type="xs:integer"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

样本 XML 文档

以下 XML 文档将由样本 XML 模式进行验证。此文档不符合某些模式规则。Age 元素的值不是数字，并且在此模式中未将 Notes 元素定义为 Person 元素的子元素。

```
<?xml version="1.0"?>
<Person xmlns="http://my.simpletest"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://my.simpletest http://my.simpletest/simple">
  <Name>
   <FirstName>Thomas</FirstName>
   <LastName>Watson</LastName>
  </Name>
  <Age>30x</Age>
  <Notes/>
<Person>
```

用来注册 XML 模式的命令

在使用 XSR_GET_PARSING_DIAGNOSTICS 存储过程来验证 XML 文档之前，必须在 DB2 XML 模式存储库（XSR）中注册用于验证的 XML 模式。以下 REGISTER XMLSCHEMA 命令假定模式位于 `c:\temp\simpleschema.xsd` 中，并且 SQL 模式为 USER1:

```
REGISTER XMLSCHEMA 'http://my.simpletest/simple'
   FROM 'file:///c:/temp/simpleschema.xsd'
   AS user1.simple COMPLETE
```

调用以生成详细的验证错误信息

以下调用从 CLP 使用 XSR_GET_PARSING_DIAGNOSTICS 存储过程来生成验证错误信息:

```
CALL XSR_GET_PARSING_DIAGNOSTICS(
blob('<?xml version="1.0"?>
<Person xmlns="http://my.simpletest"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://my.simpletest http://my.simpletest/simple">
<Name>
  <FirstName>Thomas</FirstName>
  <LastName>Watson</LastName>
 </Name>
 <Age>30x</Age>
 <Notes />
</Person>
'),'','','',1,?,?)@
```

详细的验证错误信息

调用 XSR_GET_PARSING_DIAGNOSTICS 存储过程将返回以下输出。**errorDialog** 参数的值是一个包含验证错误详细信息的 XML 文档。在此 XML 文档中，errText、location、lineNum 和 colNum 元素的内容标识特定错误以及发生错误的位置。从 CLP 命令行运行先前调用时，以下输出将写至标准输出:

```
输出参数的值
--------------------------
参数名称: ERRORDIALOG
参数值: <ErrorLog>
<XML_Error parser="XML4C">
  <errCode>238</errCode>
  <errDomain>http://apache.org/xml/messages/XML4CErrors</errDomain>
  <errText>Datatype error: Type:InvalidDatatypeValueException,
     Message:Value '30x' does not match regular expression facet '[+\-]?[0-9]+' .
  </errText>
  <lineNum>1</lineNum>
  <colNum>272</colNum>
  <location>/Person/Age</location>
  <schemaType>http://www.w3.org/2001/XMLSchema:integer</schemaType>
  <tokenCount>2</tokenCount>
  <token1>30x</token1>
  <token2>13</token2>
</XML_Error>
<XML_Error parser="XML4C">
  <errCode>2</errCode>
  <errDomain>http://apache.org/xml/messages/XMLValidity</errDomain>
  <errText>Unknown element 'Notes' </errText>
  <lineNum>1</lineNum>
  <colNum>282</colNum>
  <location>/Person</location>
  <schemaType>http://www.w3.org/2001/XMLSchema:integer</schemaType>
  <tokenCount>2</tokenCount>
  <token1>Notes</token1>
  <token2>37</token2>
</XML_Error>
<XML_Error parser="XML4C">
  <errCode>7</errCode>
  <errDomain>http://apache.org/xml/messages/XMLValidity</errDomain>
  <errText>Element 'Notes' is not valid for content model: '(Name,Age)'
  </errText>
  <lineNum>1</lineNum>
```

```
  <colNum>292</colNum>
  <location>/Person</location>
  <schemaType>http://www.w3.org/2001/XMLSchema:anyType</schemaType>
  <tokenCount>2</tokenCount>
  <token1>Notes</token1>
  <token2>31</token2>
</XML_Error>
<DB2_Error>
  <sqlstate>2200M</sqlstate>
  <sqlcode>-16210</sqlcode>
  <errText>
    [IBM][CLI Driver][DB2/NT] SQL16210N  XML document contained a value "30x" that
    violates a facet constraint. Reason code = "13".  SQLSTATE=2200M
    </errText>
</DB2_Error>
</ErrorLog>

Parameter Name  : ERRORCOUNT
Parameter Value : 3
返回状态 = 0
```

增强的错误消息支持的 ErrorLog XML 模式定义

ErrorLog XML 模式定义（XSD）描述在 XML 文档中产生解析和验证错误的情况下由 XSR_GET_PARSING_DIAGNOSTICS 存储过程生成的 UTF-8 XML 文档。输出 XML 文档存储在 *errorDialog* 参数中。

ErrorLogType

XML 模式定义的根元素是 ErrorLog 并且其类型为 ErrorLogType。

XML 模式定义

```
<xs:complexType name="ErrorLogType">
  <xs:sequence>
    <xs:element name="XML_Error" type="XML_ErrorType" minOccurs="0"
          maxOccurs="unbounded"/>
    <xs:element name=name="XML_FatalError" type="XML_ErrorType" minOccurs="0"
          maxOccurs="unbounded"/>
    <xs:element name="DB2_Error"           type="DB2_ErrorType"/>
  </xs:sequence>
</xs:complexType>
```

子元素

XML_Error

XML_FatalError

类型: XML_ErrorType

使用说明:

XML_Error 或 XML_FatalError 元素包含 XML 解析器生成的错误消息。XML_Error 和 XML_FatalError 元素具有相同的 XML 模式类型。XML_FatalError 是导致 XML 解析器异常终止解析过程的错误。

```
xs:complexType name="XML_ErrorType">
 <xs:sequence>
   <xs:element name="errCode"    type="xs:int"/>
   <xs:element name="errDomain"  type="xs:string"/>
   <xs:element name="errText"    type="xs:string"/>
   <xs:element name="lineNum"    type="xs:unsignedInt"/>
   <xs:element name="colNum"     type="xs:unsignedInt"/>
   <xs:element name="location"   type="xs:string"/>
   <xs:element name="schemaType" type="xs:string"/>
```

```
          <xs:element name="tokens">
            <xs:complexType>
              <xs:sequence minOccurs="0">
                <xs:element name="token" type="xs:string" minOccurs="0"
                     maxOccurs="unbounded"/>
             </xs:sequence>
             <xs:attribute name="count" type="xs:unsignedByte"
                  use="required"/>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       <xs:attribute name="parser" type="xs:string" use="required"/>
      </xs:complexType>
```

XML_ErrorType 元素包含下列子元素:

errCode
XML 解析器返回的错误代码。

errDomain
XML 解析器返回的错误域。

errText
原始的 XML 解析器错误消息。

lineNum
发生错误的行号。

colNum
发生错误的列号。

location
位置是发生错误之前指向最后的 XML 元素的 XPath 表达式。

schemaType
最后解析的 XML 元素的 XML 模式类型。

tokens
显示报告了多少标记的数字值。

token 标记是用来生成 DB2 错误消息的字符串值。

属性

解析器（必须的）
解析器属性指定使用的底层 XML 解析器。

DB2_Error

类型: DB2_ErrorType

使用说明:

DB2_Error 元素包含 DB2 错误消息。

```
<xs:complexType name="DB2_ErrorType">
  <xs:sequence>
    <xs:element name="sqlstate" type="xs:string"/>
    <xs:element name="sqlcode"  type="xs:int"/>
    <xs:element name="errText"  type="xs:string"/>
  </xs:sequence>
 <xs:attribute name="parser" type="xs:string" use="required"/>
</xs:complexType>
```

DB2_ErrorType 元素包含下列子元素:

sqlstate
SQLSTATE

sqlcode
SQLCCODE

errText
DB2 错误消息

增强的错误消息支持的 XML 模式

XSR_GET_PARSING_DIAGNOSTICS 存储过程将生成有关在解析和验证 XML 文档期间所产生错误的详细信息。此信息作为 XML 文档生成。此模式定义存储过程的有效 XML 输出。

以下列表表示由 XSR_GET_PARSING_DIAGNOSTICS 存储过程生成的 XML 文档的 ErrorLog XML 模式。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.ibm.com/db2/XMLParser/Diagnosticsv10"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://www.ibm.com/db2/XMLParser/Diagnosticsv10"
     elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="ErrorLog">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="XML_Error" type="XML_ErrorType" minOccurs="0"
             maxOccurs="unbounded"/>
        <xs:element name="XML_FatalError" type="XML_ErrorType" minOccurs="0"/>
        <xs:element name="DB2_Error" type="DB2_ErrorType"/>
        <xs:any namespace="##any" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="XML_ErrorType">
   <xs:attribute name="parser" type="xs:string" use="required"/>
    <xs:sequence>
      <xs:element name="errCode"    type="xs:int"/>
      <xs:element name="errDomain"  type="xs:string"/>
      <xs:element name="errText"  type="xs:string"/>
      <xs:element name="lineNum"    type="xs:unsignedInt"/>
      <xs:element name="colNum"     type="xs:unsignedInt"/>
      <xs:element name="location"   type="xs:string"/>
      <xs:element name="schemaType" type="xs:string"/>
      <xs:element name="tokens">
        <xs:complexType>
          <xs:sequence minOccurs="0">
            <xs:element name="token" type="xs:string" minOccurs="0"
                 maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="count" type="xs:unsignedByte" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:any namespace="##any"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="DB2_ErrorType">
   <xs:attribute name="parser" type="xs:string" use="required"/>
    <xs:sequence>
      <xs:element name="sqlstate" type="xs:string"/>
      <xs:element name="sqlcode"  type="xs:int"/>
      <xs:element name="errText"  type="xs:string"/>
```

```
    <xs:any namespace="##any"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

在非 Unicode 数据库中使用 XML

从版本 9.5 开始，可在不使用 Unicode 代码页的数据库中存储并检索 XML 数据。

从内部来看，XML 数据始终由 DB2 数据库服务器以 Unicode 格式管理，不管数据库代码页如何都是如此。非 XML 关系数据是以数据库代码页管理的。如果 SQL 或 XQuery 语句同时涉及 XML 数据和 SQL 关系数据（例如，在进行数据类型转换或同时涉及 XML 数据类型和 SQL 数据类型的比较时），通常需要进行代码页转换。比较 XML 数据与 XML 数据不需要代码页转换，原因是两组数据已经是 UTF-8 格式。同样，比较 SQL 数据与 SQL 数据不需要代码页转换，原因是两组数据已经是数据库代码页格式。

对于涉及 XML 数据和 SQL 数据的操作，因为 Unicode 数据库对所有数据类型使用相同编码，所以已经不需要在此数据库中进行代码页转换。但是，在非 Unicode 数据库中，涉及代码页转换的操作可能会导致数据毁坏或丢失。如果正在进行转换的 XML 数据包含带有代码点的字符不在数据库代码页中，那么会发生字符替换。因此，数据类型转换或比较操作会带来意外结果，而从数据库检索到的 XML 数据可能包含不正确的值。下面会讨论避免代码页转换以便确保已存储 XML 数据和所涉及操作的完整性的不同方法。

XML 文档插入和代码页转换

每次通过具有字符数据类型（并非 FOR BIT DATA 类型的 CHAR、VARCHAR 或 CLOB 数据类型）的主变量或参数标记将 XML 数据插入到 DB2 数据库服务器中时，如果数据库代码页不同于发出请求的客户机或应用程序的代码页，那么会进行代码页转换。当插入的字符数据从数据库代码页转换为 Unicode（XML 数据的内部管理格式）时，则会发生第二种转换。

下表显示数据库与从客户机或应用程序插入的 XML 文档字符串之间的各种可能编码组合。因为客户机通过字符数据类型插入 XML 数据，所以 XML 文档编码与客户机代码页面相同。对于每种组合，将描述代码页转换的影响和 XML 文档插入期间的结果字符替换可能性。

表 2. 数据库与插入的 XML 文档字符串之间的编码方案

方案	XML 文档编码	数据库编码	代码页是否匹配?
1.	Unicode（UTF-8）	Unicode（UTF-8）	是
2.	非 Unicode	Unicode（UTF-8）	否
3.	非 Unicode	非 Unicode	是
4.	Unicode（UTF-8）	非 Unicode	否
5.	非 Unicode	非 Unicode	否

1. 在方案 1 中，XML 文档和数据库共用 Unicode 编码。插入 XML 文档时不会进行任何字符转换。以此方式插入 XML 数据始终很安全。
2. 在方案 2 中，非 Unicode XML 文档将转换为 UTF-8 以插入到 Unicode 数据库中。此过程中不会发生字符替换。以此方式插入 XML 数据始终很安全。

3. 在方案 3 中，XML 文档和数据库共用同一非 Unicode 编码。在此情况下，XML 文档只能包含数据库代码页内包含的代码点，所以不会在代码页转换期间进行任何字符替换。以此方式插入 XML 数据始终很安全。
4. 在方案 4 中，Unicode XML 文档将插入到非 Unicode 数据库中。如果 XML 文档是从 UTF-8 客户机或应用程序插入的（通过具有字符数据类型的主变量或参数标记），那么会发生代码页转换。XML 文档内在数据库代码页中没有匹配代码点的任何字符将被替换。
5. 在方案 5 中，XML 文档将插入到同时使用两种不同编码（都是 UTF-8 以外的编码）的数据库服务器中。在此情况下，像方案 4 中一样，如果 XML 文档是使用字符数据类型插入的，那么在 XML 文档包含的字符在数据库代码页中无效时会进行字符替换。

安全地将 XML 数据插入到非 Unicode 数据库中

确保 XML 数据完整性的最安全方法是使用 Unicode 数据库。但是，如果不能使用此方法，还有其他方法可阻止字符替换。以下列表描述使用或不使用 Unicode 数据库安全插入 XML 数据的各种方法:

使用 Unicode 数据库，或确保数据库和客户机使用同一编码

如第 56 页的表 2 中所述，处于下列情况下时可以一直避免 XML 数据的代码页转换问题:

- 数据库使用 Unicode
- 数据库和客户共用同一编码，使用或不使用 Unicode

避免使用具有字符数据类型的主变量或参数标记

如果不能使用 Unicode 数据库，还可通过使用类型为 XML 或任何二进制数据类型的主变量或参数标记绑定 XML 数据来避免 XML 数据的代码页转换。即，对 XML 数据指定 CHAR、VARCHAR 或 CLOB 以外的数据类型允许直接将其从客户机或应用程序代码页转换为 Unicode，从而绕过转换为数据库代码页编码这一过程。

ENABLE_XMLCHAR 配置参数允许您控制是否通过字符数据类型插入。将 ENABLE_XMLCHAR 设置为“NO”会阻止在 XML 文档插入期间使用字符数据类型，从而避免可能的字符替换并确保已存储 XML 数据的完整性。因为 BLOB 和 FOR BIT DATA 数据对象不受代码页转换影响，所以仍然允许使用这些数据类型。缺省情况下，ENABLE_XMLCHAR 设置为“YES”以便允许插入字符数据类型。

使用 Unicode 数据库时，代码页转换决不成问题，所以在此情况下 ENABLE_XMLCHAR 配置参数不起作用；不管 ENABLE_XMLCHAR 的设置如何，都可对 XML 文档插入使用字符数据类型。

对未包含在数据库代码页中的字符使用字符实体引用

如果不能避免代码页转换并且必须对 XML 数据流使用字符数据类型，最好确保 XML 文档中的所有字符在数据库代码页中具有匹配代码点。对于 XML 数据内在目标数据库中没有匹配代码点的任何字符，可使用字符实体引用来指定字符的 Unicode 代码点。字符实体引用可一直绕过代码页转换，所以会在 XML 数据中保留正确的字符。例如，字符实体引用“ > 和 > 分别是大于号（“>”）的十六进制和十进制等价项。

在非 Unicode 数据库中查询 XML 数据

与在数据库中插入 XML 数据一样，在涉及 XML 数据的查询期间确保数据完整性的最安全方法是使用 Unicode 数据库。如果不能使用此方法，那么可通过确保所有 XML 数据在数据库代码页中可表示来避免字符替换，或者通过对数据库代码页中未包含的字符使用字符实体引用来避免字符替换。

如果查询包含的 XML 内容包括在数据库代码页中不可表示的字符，那么可能会发生下列两种类型的字符替换，从而导致查询出现意外结果:

替换为缺省替换字符

将在 XML 数据中不可匹配字符的位置引入代码页的缺省替换字符。例如，如果中文字符传递到 ASCII 编码数据库（ISO-8859-1）中，那么原始字符会替换为 ASCII 代码点 0x1A，这是一个控制字符，在客户机上通常显示为问号（“?”）。将 XML 数据从数据库代码页转换为 Unicode 时，会保留替换字符。

替换为最接近的字符等价项（“叠合”）

原始输入字符会替换为目标代码页中与原始字符相似但不一定完全相同的字符。有时具有不同 Unicode 代码点的两个或更多字符会映射至数据库代码页上的单个代码点（目标代码页中的最接近字符等价项），这样一来插入到数据库中之后不同值之间的差异会丢失。此场景已在下面的示例 2 中作出说明。

示例

以下示例说明使用 UTF-8 编码的客户机或应用程序用于查询非 Unicode 数据库中的 XML 数据时转换代码页可能带来的影响。在这些示例中，假定数据库是使用代码页 ISO8859-7（希腊语）创建的。XQuery 表达式用于匹配表 T1 中存储的 XML 数据，其中已存储 XML 数据由 Unicode 希腊语西格玛字符（Σ_G）和 Unicode 算术西格玛字符（Σ_M）组成。代码点 0xD3 在 ISO8859-7 数据库中标识西格玛字符。

表 T1 是使用下列命令创建并填充的:

```
CREATE TABLE T1 (DOCID BIGINT NOT NULL, XMLCOL XML);
INSERT INTO T1 VALUES (1, XMLPARSE(
       document '<?xml version="1.0" encoding="utf-8" ?> <Specialchars>
 <sigma>ΣG</sigma>
 <summation>ΣM</summation>
 </Specialchars>'
       preserve whitespace));
```

示例 1: 成功的代码页转换（字符在数据库代码页中是可表示的）

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "ΣG"] return $test
```

此表达式生成期望的结果:

```
<sigma>ΣG</sigma>
```

在此情况下，表达式 Σ_G 在客户机上以希腊语西格玛字符（U+03A3）的 Unicode 代码点开头，转换为希腊语数据库代码页（0xD3）中的西格玛字符，然后再转换回正确的 Unicode 字符以进行 XML 处理。因为希腊语西格玛字符在数据库代码页中是可表示的，所以表达式会正确匹配。此字符转换显示在下表中:

表 3. 字符数据转换（示例 1）

	客户机（UTF-8）		数据库（ISO8859-7）		XML 解析器（UTF-8）
字符	U+03A3（希腊语西格玛）	→	0xD3（希腊语西格玛）	→	U+03A3（希腊语西格玛）

示例 2：不成功的代码页转换（字符在数据库代码页中是不可表示的）

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "Σ_M"] return $test
```

此表达式未生成期望的结果:

```
<sigma>Σ_G</sigma>
```

在此情况下，表达式 Σ_M 在客户机上以算术符号西格玛（U+2211）的 Unicode 代码点开头，转换为希腊语数据库代码页（0xD3）中的西格玛字符，然后在进行 XML 比较时与 Σ_G 字符相匹配。对于返回表达式，过程与示例 1 中完全相同。Unicode XML 字符 Σ_G 先转换为希腊语数据库代码页中的西格玛字符（Σ_A），然后转换为客户机 UTF-8 代码页（Σ_G）中的希腊语西格玛字符。此字符转换显示在下表中:

表 4. 字符数据转换（示例 2）

	客户机（UTF-8）		数据库（ISO8859-7）		XML 解析器（UTF-8）
字符	U+2211（算术西格玛）	→	0xD3（希腊语西格玛）	→	U+03A3（希腊语西格玛）

示例 3：使用字符实体引用绕过代码页转换

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "&#2211;"]
   return $test
```

此表达式生成期望的结果:

```
<summation>Σ_M</summation>
```

在此情况下，表达式 Σ_M 在客户机上以算术符号西格玛（U+2211）开头，因为它转义为字符引用 &#2211，所以在传递至 XML 解析器时会保留 Unicode 代码点，从而能够对已存储 XML 值 Σ_M 进行成功的比较。绕过字符转换的过程显示在下表中:

表 5. 字符数据转换（示例 3）

	客户机（UTF-8）		数据库（ISO8859-7）		XML 解析器（UTF-8）
字符	U+2211（算术西格玛的字符引用）	→	″&#2211″（算术西格玛的字符引用）	→	U+2211（算术西格玛）

示例 4：不成功的代码页转换（字符在数据库代码页中是不可表示的）

此示例与示例 1 相似，但此处使用 ASCII 编码数据库并且会在 XML 表达式中引入代码页的缺省替换字符。

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "Σ_G"] return $test
```

此查询未能匹配表 T1 中的正确值。在此情况下，Unicode 字符 U+2211（希腊语西格玛）在 ASCII 代码页中没有匹配代码点，所以会引入缺省替换字符，在

此情况下为问号（“?”）。此字符转换显示在下表中:

表 6. 字符数据转换（示例 4）

	客户机（UTF-8）		数据库（ISO8859-1）		XML 解析器（UTF-8）
字符	U+2211（算术西格玛）	→	0x003F（“?”）	→	0x003F（“?”）

第 5 章 查询 XML 数据

可通过两种主查询语言（单独使用每种语言或同时使用两种语言）查询或检索存储在数据库中的 XML 数据。

为您提供了下列选项:

- 仅 XQuery 表达式
- 调用 SQL 语句的 XQuery 表达式
- 仅 SQL 语句
- 执行 XQuery 表达式的 SQL 语句

这些方法允许您从 SQL 或 XQuery 上下文中查询或检索 XML 和其他关系数据。

可以使用这些方法查询和检索 XML 文档片段或整个 XML 文档。查询可以返回片段或整个 XML 文档，并且可使用谓词限制从查询访问的结果。因为对 XML 数据的查询返回 XML 序列，所以也可以使用 XML 数据构造的查询结果。

XQuery 简介

XQuery 是万维网联盟（W3C）设计的一种函数编程语言，用于满足查询和修改 XML 数据的特定需求。

与可预测的并具有常规结构的关系数据不同，XML 数据可变性很高。XML 数据通常是不可预测、稀疏和自描述的。

由于 XML 数据的结构不可预测，所以需要对 XML 数据执行的查询通常与典型的关系查询不同。XQuery 语言提供了执行这些类型操作所需的灵活性。例如，可能需要使用 XQuery 语言执行下列操作:

- 搜索层次结构中某些未知层对象的 XML 数据。
- 对数据执行结构变换（例如，您可能想倒转层次结构）。
- 返回具有混合类型的结果。
- 更新现有 XML 数据。

XQuery 查询的组成部分

在 XQuery 中，表达式是查询的主要构建块。表达式可以进行嵌套，用来组成查询的主体。查询还可以在此主体前面具有序言。*序言*包含一系列用于定义查询处理环境的声明。*查询主体*包含用于定义查询结果的表达式。此表达式可由使用运算符或关键字组合而成的多个 XQuery 表达式构成。

第 62 页的图 4 说明了典型查询的结构。在此示例中，序言包含两个声明: 一个是版本声明，它指定要用来处理查询的 XQuery 语法的版本; 另一个是缺省名称空间声明，它指定要用于无前缀元素名称和类型名的名称空间 URI。查询主体包含一个用于构造 `price_list` 元素的表达式。`price_list` 元素的内容是将 `product` 元素按价格降序排列获得的列表。

```
Xquery version "1.0";
declare default element namespace "http://posample.org";
```

Prolog

```
<price list>{for $prod in db2-fn:xmlcolmn("PRODUCT.DESCRIPTION")/oroduct/description
        order by xs:decimal($prod/price) descending
        return <product>{$prod/name, $prod/price}</product>}
</price_list>
```

Query body

图 4. XQuery 中的典型查询的结构

使用 XQuery 函数检索 DB2 数据

在 XQuery 中，查询可以调用下列函数之一来获取 DB2 数据库中的输入 XML 数据: db2-fn:sqlquery 和 db2-fn:xmlcolumn。

db2-fn:xmlcolumn 函数将检索整个 XML 列，而 db2-fn:sqlquery 将检索基于 SQL 全查询的 XML 值。

db2-fn:xmlcolumn

db2-fn:xmlcolumn 函数采用一个字符串文字自变量，用于标识一个表或视图中的 XML 列，并返回该列中的 XML 值序列。此函数的自变量是区分大小写的。字符串文字自变量必须是类型为 XML 的限定列名。此函数允许您抽取整个 XML 数据列而不应用搜索条件。

在以下示例中，查询使用 db2-fn:xmlcolumn 函数来获取 BUSINESS.ORDERS 表的 PURCHASE_ORDER 列中的所有采购订单。然后，查询将对此输入数据执行操作，从这些采购订单的交货地址中抽取城市。查询结果是交付订单的所有城市的列表。

```
db2-fn:xmlcolumn('BUSINESS.ORDERS.PURCHASE_ORDER')/shipping_address/city
```

db2-fn:sqlquery

db2-fn:sqlquery 函数采用一个表示全查询的字符串自变量，并返回一个由全查询返回的 XML 值并置而成的 XML 序列。全查询必须指定单列结果集，而且列的数据类型必须为 XML。通过指定全查询，将允许您使用 SQL 的功能来向 XQuery 提供 XML 数据。该函数支持使用参数向 SQL 语句传递值。

在以下示例中，BUSINESS.ORDERS 表中包含名为 PURCHASE_ORDER 的 XML 列。在该示例中，查询使用 db2-fn:sqlquery 函数来调用 SQL，以获取交货日期为 2005 年 6 月 15 日的所有采购订单。然后，查询将对此输入数据执行操作，从这些采购订单的交货地址中抽取城市。查询结果是在 6 月 15 日交付订单的所有城市的列表。

```
db2-fn:sqlquery("
SELECT purchase_order FROM business.orders
WHERE ship_date = '2005-06-15' ")/shipping_address/city
```

要点: 由 db2-fn:sqlquery 或 db2-fn:xmlcolumn 函数返回的 XML 序列可以包含任何 XML 值（包括原子值和节点）。这些函数并不会总是返回格式良好的文档序列。例如，这些函数可能只返回单个原子值（例如，36）作为 XML 数据类型的实例。

对于名称是否区分大小写，SQL 和 XQuery 有不同的约定。当使用 db2-fn:sqlquery 和 db2-fn:xmlcolumn 函数时，您应知道这些差别。

SQL 是一种不区分大小写的语言
: 缺省情况下，SQL 语句中使用的所有普通标识符会自动转换为大写。因此，SQL 表和列的名称通常采用大写名称，例如，前面示例中的 BUSINESS.ORDERS 和 PURCHASE_ORDER。在 SQL 语句中，可以使用小写名称来引用这些列（例如，business.orders 和 purchase_order），在处理 SQL 语句时这些名称会自动转换为大写。（在 SQL 中，还可以通过将名称用双引号引起来创建称为*定界标识*的区分大小写的名称）。

XQuery 是一种区分大小写的语言
: XQuery 不会将小写名称转换为大写名称。这种差别会在同时使用 XQuery 和 SQL 时导致混淆。传递给 db2-fn:sqlquery 的字符串被解释为 SQL 查询并由 SQL 解析器进行解析，这会将所有名称都转换为大写。因此，在 db2-fn:sqlquery 示例中，表名 business.orders 以及列名 purchase_order 和 ship_date 既可以大写形式也可以小写形式出现。但是，db2-fn:xmlcolumn 的操作数不是 SQL 查询。操作数是区分大小写的、用于表示列名的 XQuery 字符串文字。因为实际列名是 BUSINESS.ORDERS.PURCHASE_ORDER，所以在 db2-fn:xmlcolumn 的操作数中必须用大写指定此名称。

使用 SQL 查询 XML 数据简介

可以使用 SQL 全查询或 XMLQUERY 和 XMLTABLE 的 SQL/XML 查询函数来查询 XML 数据。还可以在针对 XML 数据的 SQL 查询中使用 XMLEXISTS 谓词。

仅使用 SQL（不使用任何 XQuery）查询 XML 数据时，只能通过发出全查询在列级别进行查询。为此，只能从查询返回整个 XML 文档；仅使用 SQL 不可能返回文档片段。

要在 XML 文档内查询，需要使用 XQuery。可以使用下列任何 SQL/XML 函数或谓词通过 SQL 调用 XQuery:

XMLQUERY
: 返回 XQuery 表达式的结果作为 XML 序列的 SQL 标量函数。

XMLTABLE
: 返回 XQuery 表达式的结果作为表的 SQL 表函数。

XMLEXISTS
: 确定 XQuery 表达式是否返回非空序列的 SQL 谓词。

XQuery 与 SQL 的比较

DB2 支持将格式良好的 XML 数据存储在表列中，以及使用 SQL 和/或 XQuery 来从数据库中检索 XML 数据。支持将这两种语言用作主要查询语言，并且这两种语言都具有调用其他语言的功能。

XQuery

直接调用 XQuery 的查询以关键字 XQUERY 开头。此关键字指示正在使用 XQuery，因此 DB2 服务器必须使用适用于 XQuery 语言且区分大小写的规则。错误处理是根据用来处理 XQuery 表达式的接口来进行的。报告 XQuery 错误时会提供 SQLCODE 和 SQLSTATE，与报告 SQL 错误的方式相同。处理 XQuery 表达式时不会返回警告。XQuery 通过调用从 DB2 表和视图中抽取 XML 数据的函数来获取数据。也可从 SQL 查询中调用 XQuery。在此情况下，SQL 查询可通过绑定变量的方式将 XML 数据传递给 XQuery。XQuery 支持用来处理 XML 数据和构造新 XML 对象（例如，元素和属性）的各种表达式。用于 XQuery 的编程接口所提供的功能与 SQL 的功能类似，可用于准备查询和检索查询结果。

SQL

SQL 能够定义和实例化 XML 数据类型的值。包含格式良好的 XML 文档的字符串可解析为 XML 值，（可选）针对 XML 模式进行验证，在表中插入或更新。或者，可使用 SQL 构造函数来构造 XML 值；在此过程中会将其他关系数据转换为 XML 值。还提供了一些函数，使得可使用 XQuery 来查询 XML 数据以及将 XML 数据转换为关系表以用于 SQL 查询。除了可将 XML 值序列化为字符串数据以外，还可以在 SQL 数据类型与 XML 数据类型之间进行转换。

SQL/XML 提供了下列函数和谓词，以便从 SQL 来调用 XQuery:

XMLQUERY

XMLQUERY 是一个标量函数，它将 XQuery 表达式作为自变量，返回的是一个 XML 序列。该函数有一些可选参数，可用来将 SQL 值作为 XQuery 变量传递给 XQuery 表达式。可以在 SQL 查询的上下文中进一步处理由 XMLQUERY 返回的 XML 值。

XMLTABLE

XMLTABLE 是一个表函数，它使用 XQuery 表达式来从 XML 数据生成 SQL 表，SQL 可以进一步处理该 SQL 表。

XMLEXISTS

XMLEXISTS 是一个 SQL 谓词，它确定 XQuery 表达式是否返回由一个或多个项组成的序列（而不是一个空序列）。

比较用于查询 XML 数据的方法

因为可以通过使用 XQuery、SQL 或 XQuery 和 SQL 的组合来用许多方法查询 XML 数据，所以要选择的方法随情况不同而变化。下列各节描述对特定查询方法有利的条件。

仅 XQuery

在下列情况下，适合只使用 XQuery 进行查询:

- 应用程序仅访问 XML 数据，而不需要查询非 XML 关系数据

- 将先前用 XQuery 编写的查询迁移到 DB2 数据库 Linux® 版、UNIX® 版和 Windows® 版
- 返回将用作值的查询结果以构造 XML 文档
- 与 SQL 相比，查询发起人更熟悉 XQuery

调用 SQL 的 XQuery

在下列情况下（上一节中确定的仅使用 XQuery 的情况除外），适合使用调用 SQL 的 XQuery 进行查询:

- 查询涉及 XML 数据和关系数据；可以在查询中利用对关系列定义的 SQL 谓词和索引
- 要将 XQuery 表达式应用于下列各项的结果:
  - UDF 调用，因为不能直接通过 XQuery 调用这些 UDF
  - 使用 SQL/XML 发布函数通过关系数据构造的 XML 值
  - 使用 DB2 Net Search Extender 的查询，提供 XML 文档的全文本搜索但必须与 SQL 配合使用

仅 SQL

仅使用 SQL（不使用任何 XQuery）检索 XML 数据时，只能在 XML 列级别进行查询。为此，只能从查询返回整个 XML 文档。此用法在下列情况下适用:

- 需要检索整个 XML 文档
- 不需要根据存储文档内的值进行查询，或者查询的谓词在表的其他非 XML 列中

执行 XQuery 表达式的 SQL/XML 函数

SQL/XML 函数 XMLQUERY 和 XMLTABLE 以及 XMLEXISTS 谓词使 XQuery 表达式能够在 SQL 上下文中执行。在下列情况下，适合在 SQL 内执行 XQuery:

- 需要启用现有 SQL 应用程序以在 XML 文档内查询。要在 XML 文档内查询，需要执行 XQuery 表达式，这可通过使用 SQL/XML 来完成
- 查询 XML 数据的应用程序需要将参数标记传递至 XQuery 表达式。（参数标记首先绑定至 XMLQUERY 或 XMLTABLE 中的 XQuery 变量。）
- 与 XQuery 相比，查询发起人更熟悉 SQL
- 需要在单次查询中返回关系数据和 XML 数据
- 需要连接 XML 和关系数据
- 需要分组或聚集 XML 数据。可以将子查询的 GROUP BY 或 ORDER BY 子句应用于 XML 数据（例如，使用 XMLTABLE 函数以表格式检索和收集 XML 数据之后）

指定 XML 名称空间

在 XML 文档中，XML 名称空间是可选的，用作 XML 文档中节点名的前缀。要访问使用名称空间的 XML 文档中的节点，XQuery 表达式必须同时指定同一名称空间作为节点名的一部分。可对文档指定缺省 XML 名称空间，并且可对文档中的特定元素指定 XML 名称空间。

注意，名称空间声明用分号（;）结束。这意味着，如果还想使用包含分号的 SQL 语句和 XQuery 表达式，那么不能使用分号作为语句终止字符（例如，通过使用 db2 -t 调用命令行处理器）。还可使用 -td 选项指定分号以外的终止字符，这可以确保不会错误地解释包含名称空间声明的语句。教程中的示例使用颚化音（~）作为终止字符（-td~），但也常常使用 %（-td%）。

例如，pureXML 的教程使用对 XML 文档指定了缺省元素名称空间的 XML 文档。以下 XML 是教程中使用的其中一个 XML 文档:

```
<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>
```

XML 文档的根节点将文档的缺省元素名称空间绑定至通用资源标识（URI）http://posample.org。

```
<customerinfo xmlns="http://posample.org" Cid="1002">
```

通过包括 **declare default element namespace** 序言，您在教程中运行的 XQuery 表达式还会将 URI 作为缺省元素名称空间绑定。例如，以下 SELECT 语句中的 XQuery 表达式声明缺省元素名称空间；如果对教程中创建的 CUSTOMER 表运行 SELECT 语句，那么会返回客户标识:

```
SELECT cid FROM customer
   WHERE XMLEXISTS('declare default element namespace "http://posample.org";
   $i/customerinfo/addr/city[ . = "Markham"]' passing INFO as "i")
```

通过在 XML 文档中使用同一 URI 作为缺省元素名称空间，该表达式可使用正确的名称空间前缀在表达式中限定节点名。如果没有缺省元素名称空间声明，或者将另一 URI 绑定为缺省元素名称空间，那么表达式不会使用正确的名称空间限定节点名，并且不会返回任何数据。例如，以下 SELECT 语句类似于上一语句，但它没有缺省名称空间声明。如果对教程中创建的 CUSTOMER 表运行此语句，那么不会返回任何数据。

```
SELECT cid FROM customer
WHERE XMLEXISTS('$i/customerinfo/addr/city[ . = "Markham"]'
   passing INFO as "i")
```

将名称空间前缀与节点名配合使用

要使用名称空间限定节点名，可对每个节点名添加名称空间前缀。使用冒号隔开前缀与节点名。对于节点 po:addr，名称空间前缀 po 与本地节点名 addr 隔开。如果使用节点名来限定名称空间前缀，那么必须确保该前缀绑定至 URI。例如，以下 SELECT 语句中的 XQuery 表达式通过声明名称空间 po 将名称空间前缀 po 绑定至 URI http://posample.org。对教程中创建的 CUSTOMER 表运行以下语句时，会返回一个结果。

```
SELECT cid FROM customer
   WHERE XMLEXISTS('
    declare namespace po = "http://posample.org";
   $i/po:customerinfo/po:addr/po:city[ . = "Markham"]' passing INFO as "i")
```

名称空间前缀 po 可能是任何前缀；重要的是绑定至前缀的 URI。例如，以下 SELECT 语句中的 XQuery 表达式使用名称空间前缀 mytest，但相当于上一语句中的表达式:

```
SELECT cid FROM customer
WHERE XMLEXISTS('declare namespace mytest = "http://posample.org";
   $i/mytest:customerinfo/mytest:addr/mytest:city[ . = "Markham"]'
   passing INFO as "i")
```

使用通配符作为名称空间前缀

可在 XQuery 表达式中使用通配符来匹配 XML 数据中使用的任何名称空间。以下 SELECT 语句中的 XQuery 表达式使用通配符来匹配所有名称空间前缀。

```
SELECT cid FROM customer
WHERE XMLEXISTS('$i/*:customerinfo/*:addr/*:city[ . = "Markham"]'
   passing INFO as "i")
```

对教程中创建的 CUSTOMER 表运行 SELECT 语句时，会返回一个客户标识。

示例：更改元素的名称空间前缀

这些示例说明了如何添加和除去为元素的 QName 指定的名称空间绑定。

元素的 QName 是可选名称空间前缀和局部名。名称空间前缀与局部名用冒号隔开。如果名称空间前缀存在，那么会绑定至 URI（通用资源标识），并且会提供 URI 的简短形式。扩展 QName 包括名称空间 URI 和局部名。

通过使用诸如 fn:QName、fn:local-name 和 fn:namespace-uri 等函数，可以重命名属性和节点名以及查找与节点的名称空间前缀相关联的 URI。

XQuery 名称空间声明以分号（;）结尾，但不能使用分号作为语句的终止字符。这些示例使用颚化音（~）作为终止字符，与 pureXML 教程中使用的终止字符相同。

对元素添加名称空间前缀

这些示例使用表中的一个 XML 文档。下列语句将创建此表并将 XML 文档插入此表中:

```
CREATE TABLE XMLTEST (ID BIGINT NOT NULL PRIMARY KEY, XMLDOC XML ) ~
INSERT INTO XMLTEST Values (4,
  '<depts>
    <dept id="A07">
        <emp id="31201" >
           <location region="31" />
              </emp>
        <emp type="new" id = "23322" >
          <moved:location xmlns:moved="http://oldcompany.com" region="43" />
        </emp>
    </dept>
    </depts> ') ~
```

以下 SELECT 语句中的 XQuery 表达式将向元素添加名称空间前缀。XQuery 表达式使用 fn:QName 来创建一个具有名称空间绑定的 QName。

XQuery let 子句将创建一个名称为 emp 并且名称空间为 http://example.com/new 的空元素。在 transform 表达式中，rename 表达式将更改 XPath 表达式 $newdept/depts/dept/emp[@type="new"] 所标识的元素的名称。此元素的名称为 emp，但是没有名称空间前缀。

```
SELECT XMLQUERY ( '
  let $newemp := fn:QName( "http://mycompany.com", "new:emp")
  return
```

```
  transform
    copy $newdept := $doc
    modify
    do rename $newdept/depts/dept/emp[@type="new"] as $newemp
  return
  $newdept  ' passing XMLDOC as "doc" )
from XMLTEST where ID = 4 ~
```

XQuery 表达式将返回以下 XML 数据以及指定为 new:emp 节点的一部分的名称空间绑定。经过修改的文档是一个有效 XML，它包含已绑定至名称空间的名称空间前缀。已将名称空间声明添加至已重命名的新元素。

```
<depts>
 <dept id="A07">
  <emp id="31201">
   <location region="31" />
  </emp>
  <new:emp xmlns:new="http://mycompany.com" type="new" id="23322">
   <moved:location xmlns:moved="http://oldcompany.com" region="43" />
  </new:emp>
     </dept>
</depts>
```

从元素中除去名称空间前缀

可以通过将节点重命名为不具有名称空间绑定的 QName，来实现从节点名中除去名称空间前缀。SELECT 语句中的以下 XQuery 表达式使用一个 for 子句和 fn:namespace-uri 函数来确定每个 emp 节点中元素节点的 URI。如果 URI 为 http://oldcompany.com，那么 rename 表达式将使用 fn:QName 和 fn:local-name 函数从元素节点中除去名称空间前缀。

```
SELECT XMLQUERY ( '
  transform
    copy $newdept := $x
    modify
     for $testemp in $newdept/depts/dept/*:emp/*
     return
     if ( fn:namespace-uri( $testemp ) eq "http://oldcompany.com")
     then
       do rename $testemp as fn:QName( "", fn:local-name($testemp) )
     else()
  return
  $newdept
 ' passing XMLDOC as "x" )
from XMLTEST where ID = 4 ~
```

XQuery 表达式将返回以下 XML 数据。节点名 new:location 被替换为 location。未从节点中除去名称空间绑定。

```
<depts>
  <dept id="A07">
    <emp id="31201">
      <location region="31" />
          </emp>
    <emp type="new" id="23322">
      <location xmlns:moved="http://oldcompany.com" region="43" />
    </emp>
  </dept>
</depts>
```

XMLQUERY 函数概述

XMLQUERY 是一个 SQL 标量函数，它允许您在 SQL 上下文中执行 XQuery 表达式。可以将变量传递至在 XMLQUERY 中指定的 XQuery 表达式。XMLQUERY 返回 XML 值，它是 XML 序列。此序列可以为空，也可以包含一项或多项。

通过在 SQL 上下文中执行 XQuery 表达式，可以完成下列任务:

- 对存储的部分 XML 文档而不是整个 XML 文档进行操作（只有 XQuery 可以在 XML 文档内查询；仅 SQL 时在整个文档级别查询）
- 使 XML 数据能够参与 SQL 查询
- 同时对关系数据和 XML 数据进行操作
- 将进一步的 SQL 处理应用于返回的 XML 值（例如，使用子查询的 ORDER BY 子句对结果进行排序）

有关更多详细信息，请参阅有关比较查询方法的文档。

请注意，XQuery 区分大小写，因此指定在 XMLQUERY 中指定的 XQuery 表达式和变量时必须小心。

不需要传递 SQL 表达式的完整功能时，还提供了用于传递列名而不必在 **passing** 子句中显式指定名称的较简单语法。请参阅使用 XMLEXISTS、XMLQUERY 和 XMLTABLE 进行简单列名传递。

XMLQUERY 返回的非空序列

如果其中指定的 XQuery 表达式产生非空序列，那么 XMLQUERY 函数返回非空序列。

例如，如果下列两个 XML 文档存储在 CUSTOMER 表的 XML 列 INFO 中:

```
<customerinfo Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
    <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

如果发出以下查询:

```
SELECT XMLQUERY ('$d/customerinfo/phone' passing INFO as "d")
FROM CUSTOMER
```

那么生成的表将包含如下两行（已对该表进行格式编排以便更加清楚明了）:

表 7. 结果表

<phone type="work">905-555-7258</phone>
<phone type="work">905-555-7258</phone><phone type="home">416-555-2937</phone><phone type="cell">905-555-8743</phone><phone type="cottage">613-555-3278</phone>

请注意，第一行包含一个 <phone> 元素的序列，而第二行包含四个 <phone> 元素的序列。生成此结果是因为第二个 XML 文档包含四个 <phone> 元素，并且 XMLQUERY 返回满足 XQuery 表达式的所有元素的序列。（请注意，第二行中的结果是结构不当的文档。请确保接收此结果的任何应用程序都能正确处理此行为。）

上一个示例说明了通常使用 XMLQUERY 的方式：一次应用于一个 XML 文档，其中生成的表的每行表示一个文档产生的结果。但是，还可以将 XMLQUERY 同时应用于多个文档，就像多个文档包含在单个序列中一样。在此示例中，将 XMLQUERY 应用于序列中的所有文档产生的结果返回在一行中。

例如，假定上面显示的那个文档存储在 CUSTOMER 表的 INFO 列中。以下查询中的 db2-fn:xmlcolumn 函数返回一个在 INFO 列中包含两个 XML 文档的序列。

```
VALUES
 (XMLQUERY
   ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo/phone'))
```

然后，将 XMLQUERY 应用于 XML 文档的这一个序列，生成的表将只包含一行，如下所示:

表 8. 结果表

<phone type="work">905-555-7258</phone><phone type="work">905-555-7258</phone><phone type="home">416-555-2937</phone><phone type="cell">905-555-8743</phone><phone type="cottage">613-555-3278</phone>

INFO 列中 XML 文档中的所有 <phone> 元素都返回在一行中，因为 XMLQUERY 对单个值进行操作：从 db2-fn:xmlcolumn 返回的 XML 文档的序列。

XMLQUERY 返回的空序列

如果其中指定的 XQuery 表达式返回空序列，那么 XMLQUERY 函数返回空序列。

例如，在以下查询中，对于 CUSTOMER 表的 INFO 列中不包含值为“Aurora”的 <city> 元素的每一行，XMLQUERY 将返回空序列。

```
SELECT Cid, XMLQUERY ('$d//addr[city="Aurora"]' passing INFO as "d") AS ADDRESS
FROM CUSTOMER
```

假定 CUSTOMER 表有三行，但只有一个 XML 文档包含值为“Aurora”的 <city> 元素。上一个 SELECT 语句将产生下表（已对该输出进行格式编排以便更加清楚明了）。

表 9. 结果表

CID	ADDRESS
1001	
1002	

表 9. 结果表 (续)

CID	ADDRESS
1003	<addr country="Canada"><street>1596 Baseline</street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-zip>N8X-7F8</pcode-zip></addr>

请注意是如何对不包含值为“Aurora”的 <city> 元素的行返回长度为零的已序列化 XML，而不是空值的空序列。但是，将在第三行中返回 <addr> 元素，因为它满足 XQuery 表达式。在第三行中，将返回一个非空序列。

通过在语句的 WHERE 子句而不是 SELECT 子句中应用谓词（如 XMLEXISTS），可以避免返回包含空序列的行。例如，可以按如下所示编写上一个查询，将 XMLQUERY 函数中的过滤谓词移动到 WHERE 子句:

```
SELECT Cid, XMLQUERY ('$d/customerinfo/addr' passing c.INFO as "d")
FROM Customer as c
WHERE XMLEXISTS ('$d//addr[city="Aurora"]' passing c.INFO as "d")
```

通过此查询生成的表如下所示:

表 10. 结果表

CID	ADDRESS
1003	<addr country="Canada"><street>1596 Baseline</street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-zip>N8X-7F8</pcode-zip></addr>

XMLQUERY 通常用在 SELECT 子句中以返回所选文档的片段。在 XMLQUERY 的 XQuery 表达式中指定的谓词不过滤结果集中的行，它们仅用于确定返回的片段。要真正除去结果集中的行，需要在 WHERE 子句中应用谓词。XMLEXISTS 谓词可用来应用依赖于存储的 XML 文档内的值的谓词。

将 XMLQUERY 结果的数据类型转换为非 XML 类型

如果想要将 XMLQUERY 函数的结果返回至 SQL 上下文以进行进一步处理（如进行比较或排序操作），那么需要将返回的 XML 值转换为兼容的 SQL 类型。指定 XMLCAST 将使您能够在 XML 值和非 XML 值之间进行转换。

注:

1. 仅当在 XMLQUERY 中指定的 XQuery 表达式返回的序列包含一个已打散的项时，才能将 XMLQUERY 的结果转换为 SQL 数据类型。
2. 在非 UTF-8 数据库中，将 XMLQUERY 的结果转换为 SQL 数据类型会导致代码页转换，原因是返回的值会从内部 UTF-8 编码转换为数据库代码页。返回值内不在数据库代码页中的所有代码点将替换为替换字符。引入替换字符会导致 XML 与非 XML 值之间的比较出现意外行为，所以应注意确保已存储的 XML 数据仅包含数据库代码页内包括的代码点。

示例：比较查询中的 XML 值和非 XML 值

在以下查询中，将 XMLQUERY 返回的序列从其 XML 类型转换为字符类型，以便可以将它与 PRODUCT 表的 NAME 列比较。（如果从 XMLQUERY 返回的 XML 值不是序列化的字符串，那么 XMLCAST 操作将失败。）

```
SELECT R.Pid
FROM PURCHASEORDER P, PRODUCT R
WHERE R.NAME =
        XMLCAST( XMLQUERY ('for $pname in $d/PurchaseOrder/item/name
                               return $pname'
                            PASSING P.PORDER AS "d") AS VARCHAR(128))
```

示例：按 XMLQUERY 结果进行排序

在以下查询中，按产品描述的 <name> 元素的值排序的顺序返回产品标识，该产品描述作为 XML 文档存储。因为 SQL 不能对 XML 值进行排序，所以必须将序列转换为 SQL 可对其进行排序的值（在此示例中，为字符）。

```
SELECT Pid
FROM PRODUCT
ORDER BY XMLCAST(XMLQUERY ('$d/product/description/name'
                            PASSING DESCRIPTION AS "d") AS VARCHAR(128))
```

数据类型之间的强制转换

在许多情况下，需要将具有给定数据类型的值*强制类型转换*为另一种数据类型，或者转换为长度、精度或小数位不同的同一数据类型。数据类型提升是一个示例，将一种数据类型提升为另一种数据类型时需要将值强制转换为新的数据类型。可强制转换为另一种数据类型的一种数据类型可从源数据类型*强制类型转换*为目标数据类型。

可以将一种数据类型隐式或显式地强制转换为另一种数据类型。根据涉及到的数据类型，可以使用强制类型转换函数、CAST 规范或 XMLCAST 规范来显式更改数据类型。另外，当创建了用户定义的有源函数时，必须能够将源函数的参数的数据类型强制转换为正在创建的函数的数据类型。

第 74 页的表 11 中显示了内置数据类型之间受支持的强制类型转换。第一列表示强制类型转换操作数的数据类型（源数据类型），而顶部的数据类型表示强制类型转换操作的目标数据类型。“Y”表示可以将 CAST 规范用于源数据类型和目标数据类型的组合。说明了只能使用 XMLCAST 规范的那些情况。

如果将任何数据类型强制转换为字符或图形数据类型时任何非空白字符被截断，就会返回警告。此截断行为与对字符或图形数据类型赋值时任何非空白字符被截断而发生错误的情况不同。

涉及到单值类型的下列强制类型转换是受支持的（除非另有声明，否则都使用 CAST 规范来进行强制类型转换）：

- 从单值类型 *DT* 强制转换为它的源数据类型 *S*
- 从单值类型 *DT* 的源数据类型 *S* 强制转换为单值类型 *DT*
- 从单值类型 *DT* 强制转换为同一单值类型 *DT*
- 从数据类型 *A* 强制转换为单值类型 *DT*；其中，可将 *A* 提升为单值类型 *DT* 的源数据类型 *S*
- 从 INTEGER 强制转换为一种源数据类型为 SMALLINT 的单值类型 *DT*

- 从 DOUBLE 强制转换为一种源数据类型为 REAL 的单值类型 *DT*
- 从 DECFLOAT 强制转换为源数据类型为 DEFLOAT 的单值类型 DT
- 从 VARCHAR 强制转换为一种源数据类型为 CHAR 的单值类型 *DT*
- 从 VARGRAPHIC 强制转换为一种源数据类型为 GRAPHIC 的单值类型 *DT*
- 对于 Unicode 数据库，从 VARCHAR 或 VARGRAPHIC 强制转换为一种源数据类型为 CHAR 或 GRAPHIC 的单值类型 *DT*
- 使用 XMLCAST 规范从一种具有源数据类型 *S* 的单值类型 *DT* 强制转换为 XML
- 根据 XML 值的 XML 模式数据类型，使用 XMLCAST 规范从 XML 强制转换为具有可为任何内置数据类型的源数据类型的单值类型 *DT*

不能将 FOR BIT DATA 字符类型强制转换为 CLOB。

对于涉及到将数组类型作为目标的强制类型转换，必须能够将源数组的元素的数据类型强制转换为目标数组数据的元素的数据类型（SQLSTATE 42846）。如果目标数组类型是一个普通数组，那么源数组值必须是一个普通数组（SQLSTATE 42821），并且源数组值的基数必须小于或等于目标数组数据类型的最大基数（SQLSTATE 2202F）。如果目标数组类型是一个联合数组，那么必须能够将源数组值的索引的数据类型强制转换为目标数组类型的索引的数据类型。只能将用户定义的数组类型值强制转换为用户定义的另一种同名的数组类型（SQLSTATE 42846）。

不能将游标类型作为 CAST 规范的源数据类型或目标数据类型，除非将参数标记强制转换为游标类型。

对于涉及到将行类型作为目标的强制类型转换，源行值表达式的等级必须与目标行类型的等级相匹配，并且必须能够将源行值表达式中的每个字段强制转换为相应的目标字段。只能将用户定义的行类型值强制转换为用户定义的另一种同名的行类型（SQLSTATE 42846）。

无法将结构化类型值强制转换为其他类型值。不需要将结构化类型 *ST* 强制转换为它的其中一种超类型，这是因为 *ST* 的超类型的所有方法都适用于 *ST*。如果期望的操作只适用于 *ST* 的子类型，那么使用子类型处理表达式来将 *ST* 当作它的一种子类型来处理。

当未使用模式名来对强制类型转换中涉及到的用户定义的数据类型加以限定时，使用 *SQL 路径*来查找包括该用户定义的数据类型的具有该名称的第一种模式。

涉及到引用类型的下列强制类型转换是受支持的:

- 从引用类型 *RT* 强制转换为它的表示数据类型 *S*
- 从引用类型 *RT* 的表示数据类型 *S* 强制转换为引用类型 *RT*
- 从目标类型为 *T* 的引用类型 *RT* 强制转换为目标类型为 *S* 的引用类型 *RS*，其中 *S* 是 *T* 的超类型。
- 从数据类型 *A* 强制转换为引用类型 *RT*，其中，可将 *A* 提升为引用类型 *RT* 的表示数据类型 *S*。

当未使用模式名来对强制类型转换中涉及到的参考数据类型的目标类型加以限定时，使用 *SQL 路径*来查找包括该用户定义的数据类型的具有该名称的第一种模式。

表 11. 受支持的内置数据类型之间的强制类型转换

	目标数据类型																				
源数据类型	SMALLINT	INTEGER	BIGINT	DECIMAL	REAL	DOUBLE	DECFLOAT	CHAR	CHAR FBD[2]	VARCHAR	VARCHAR FBD[2]	CLOB	GRAPHIC	VARGRAPHIC	DBCLOB	BLOB	DATE	TIME	TIMESTAMP	XML	BOOLEAN
SMALLINT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	-
INTEGER	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	-
BIGINT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	-
DECIMAL	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	-
REAL	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	-
DOUBLE	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	-
DECFLOAT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	-	-
CHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y[1]	Y[1]	Y[1]	Y	Y	Y	Y	Y[4]	-
CHAR FOR BIT DATA	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	Y	Y	Y	Y	Y[3]	-
VARCHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y[1]	Y[1]	Y[1]	Y	Y	Y	Y	Y[4]	-
VARCHAR FOR BIT DATA	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	Y	Y	Y	Y	Y[3]	-
CLOB	-	-	-	-	-	-	-	Y	-	Y	-	Y	Y[1]	Y[1]	Y[1]	Y	-	-	-	Y[4]	-
GRAPHIC	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	-	Y[1]	-	Y[1]	Y	Y	Y	Y	Y[1]	Y[1]	Y[1]	Y[3]	-
VARGRAPHIC	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	-	Y[1]	-	Y[1]	Y	Y	Y	Y	Y[1]	Y[1]	Y[1]	Y[3]	-
DBCLOB	-	-	-	-	-	-	-	Y[1]	-	Y[1]	-	Y[1]	Y	Y	Y	Y	-	-	-	Y[3]	-
BLOB	-	-	-	-	-	-	-	-	Y	-	Y	-	-	-	-	Y	-	-	-	Y[4]	-
DATE	-	Y	Y	Y	-	-	-	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	Y	-	Y	Y[3]	-
TIME	-	Y	Y	Y	-	-	-	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	Y	-	Y[3]	-
TIMESTAMP	-	-	Y	Y	-	-	-	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	Y	Y	Y	Y[3]	-
XML	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y	-
BOOLEAN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

表 11. 受支持的内置数据类型之间的强制类型转换 (续)

	目标数据类型																				
源数据类型	SMALLINT	INTEGER	BIGINT	DECIMAL	REAL	DOUBLE	DECFLOAT	CHAR	CHAR FBD[2]	VARCHAR	VARCHAR FBD[2]	CLOB	GRAPHIC	VARGRAPHIC	DBCLOB	BLOB	DATE	TIME	TIMESTAMP	XML	BOOLEAN

说明

- 有关涉及到用户定义的类型和引用类型的受支持的强制类型转换的信息，请参阅该表前面的描述。
- 无法将结构化类型值强制转换为其他任何类型值。
- LONG VARCHAR 和 LONG VARGRAPHIC 数据类型继续受支持但是不推荐使用，并且在将来的发行版中可能会除去。

[1] 仅支持对 Unicode 数据库进行强制类型转换。

[2] FOR BIT DATA

[3] 只能使用 XMLCAST 来执行强制类型转换。

[4] 隐式处理 XMLPARSE 函数，以便在对 XML 列的赋值（INSERT 或 UPDATE）为字符串时将字符串转换为 XML。字符串必须是结构良好的 XML 文档，赋值才能成功。

[5] 只能使用 XMLCAST 并且根据 XML 值的底层 XML 模式数据类型来执行强制类型转换。有关详细信息，请参阅"XMLCAST"。

[6] 不能将游标类型作为 CAST 规范的源数据类型或目标数据类型，除非将参数标记强制转换为游标类型。

表 表 12 显示了在强制转换为标识的目标数据类型时应在哪里查找有关所适用的规则的信息。

表 12. 有关强制转换为某种数据类型的规则

目标数据类型	规则
SMALLINT	*SQL Reference, Volume 1*中的『SMALLINT 标量函数』
INTEGER	*SQL Reference, Volume 1*中的『INTEGER 标量函数』
BIGINT	*SQL Reference, Volume 1*中的『BIGINT 标量函数』
DECIMAL	*SQL Reference, Volume 1*中的『DECIMAL 标量函数』
NUMERIC	*SQL Reference, Volume 1*中的『DECIMAL 标量函数』
REAL	*SQL Reference, Volume 1* 中的『REAL 标量函数』

表 12. 有关强制转换为某种数据类型的规则 (续)

目标数据类型	规则
DOUBLE	*SQL Reference, Volume 1*中的『DOUBLE 标量函数』
DECFLOAT	*SQL Reference, Volume 1*中的『DECFLOAT 标量函数』
CHAR	*SQL Reference, Volume 1* 中的『CHAR 标量函数』
VARCHAR	*SQL Reference, Volume 1* 中的『VARCHAR 标量函数』
CLOB	*SQL Reference, Volume 1* 中的『CLOB 标量函数』
GRAPHIC	*SQL Reference, Volume 1* 中的『GRAPHIC 标量函数』
VARGRAPHIC	*SQL Reference, Volume 1* 中的『VARGRAPHIC 标量函数』
DBCLOB	*SQL Reference, Volume 1* 中的『DBCLOB 标量函数』
BLOB	*SQL Reference, Volume 1* 中的『BLOB 标量函数』
DATE	*SQL Reference, Volume 1* 中的『DATE 标量函数』
TIME	*SQL Reference, Volume 1* 中的『TIME 标量函数』
TIMESTAMP	如果源类型为字符串，请参阅 *SQL Reference, Volume 1* 中的『TIMESTAMP 标量函数』，其中指定了一个操作数。如果源数据类型为 DATE，那么时间戳记由指定的日期和时间 00:00:00 组成。

将非 XML 值强制转换为 XML 值

表 13. 受支持的从非 XML 值强制转换为 XML 值

	目标数据类型	
源数据类型	XML	获得的 XML 模式类型
SMALLINT	Y	xs:short
INTEGER	Y	xs:int
BIGINT	Y	xs:long
DECIMAL 或 NUMERIC	Y	xs:decimal
REAL	Y	xs:float
DOUBLE	Y	xs:double
DECFLOAT	N	-
CHAR	Y	xs:string
VARCHAR	Y	xs:string
CLOB	Y	xs:string

表 13. 受支持的从非 XML 值强制转换为 XML 值 (续)

源数据类型	目标数据类型	
	XML	**获得的 XML 模式类型**
GRAPHIC	Y	xs:string
VARGRAPHIC	Y	xs:string
DBCLOB	Y	xs:string
DATE	Y	xs:date
TIME	Y	xs:time
TIMESTAMP	Y	xs:dateTime[1]
BLOB	Y	xs:base64Binary
字符类型 FOR BIT DATA	Y	xs:base64Binary
单值类型		将此图表与单值类型的源类型配合使用

说明

[1] 源数据类型 TIMESTAMP 支持的时间戳记精度为 0 到 12。xs:dateTime 的最大小数秒数精度为 6。如果源数据类型 TIMESTAMP 的时间戳记精度超过了 6，那么值在强制转换为 xs:dateTime 时将被截断。

LONG VARCHAR 和 LONG VARGRAPHIC 数据类型继续受支持但是不推荐使用，并且在将来的发行版中可能会除去。

当将字符串值强制转换为 XML 值时，获得的 xs:string 原子值不能包含非法 XML 字符（SQLSTATE 0N002）。如果输入字符串不是采用 Unicode，那么会将输入字符转换为使用 Unicode。

对 SQL 二进制类型进行强制类型转换时将获得类型为 xs:base64Binary 的 XQuery 原子值。

将 XML 值强制转换为非 XML 值

可以认为通过 XMLCAST 将 XML 值转换为非 XML 值这一过程包含两个强制类型转换：一个是 XQuery 强制类型转换，它将源 XML 值转换为对应于 SQL 目标类型的 XQuery 类型；接着是从相应的 XQuery 类型强制转换为真正的 SQL 类型。

如果目标类型具有相应的受支持的 XQuery 目标类型，并且具有受支持的用于从源值类型转换为相应的 XQuery 目标类型的 XQuery 强制类型转换，那么 XMLCAST 是受支持的。在 XQuery 强制类型转换中使用的目标类型依赖于相应的 XQuery 目标类型，并且可能包含一些附加限制。

下表列示了通过这种转换获得的 XQuery 类型。

表 14. 受支持的从 XML 值强制转换为非 XML 值

目标数据类型	源数据类型	
	XML	**相应的 XQuery 目标类型**
SMALLINT	Y	xs:short
INTEGER	Y	xs:int
BIGINT	Y	xs:long

表 14. 受支持的从 XML 值强制转换为非 XML 值（续）

目标数据类型	源数据类型	
	XML	相应的 XQuery 目标类型
DECIMAL 或 NUMERIC	Y	xs:decimal
REAL	Y	xs:float
DOUBLE	Y	xs:double
DECFLOAT	Y	没有匹配类型[1]
CHAR	Y	xs:string
VARCHAR	Y	xs:string
CLOB	Y	xs:string
GRAPHIC	Y	xs:string
VARGRAPHIC	Y	xs:string
DBCLOB	Y	xs:string
DATE	Y	xs:date
TIME（不带时区）	Y	xs:time
TIMESTAMP（不带时区）	Y	xs:dateTime[2]
BLOB	Y	xs:base64Binary
CHAR FOR BIT DATA	N	不可执行强制类型转换
VARCHAR FOR BIT DATA	Y	xs:base64Binary
单值类型		将此图表与单值类型的源类型配合使用
行、引用、结构化或抽象数据类型（ADT），或者其他	N	不可执行强制类型转换

说明

[1] DB2 支持 XML 模式 1.0，该模式没有为 DECFLOAT 提供匹配的 XML 模式类型。按以下方式处理 XMLCAST 的 XQuery 强制类型转换步骤:

- 如果源值的类型为 XML 模式数字类型，那么使用该数字类型。
- 如果源值的类型为 XML 模式类型 xs:boolean，那么使用 xs:double。
- 否则，使用 xs:string 并执行附加检查以找到有效数字格式。

[2] xs:dateTime 的最大小数秒数精度为 6。源数据类型 TIMESTAMP 支持的时间戳记精度为 0 到 12。如果目标数据类型 TIMESTAMP 的时间戳记精度小于 6，那么值在从 xs:dateTime 强制转换时将被截断。如果目标数据类型 TIMESTAMP 的时间戳记精度超过了 6，那么值在从 xs:dateTime 强制转换时将对它填充 0。

在下列限制情况下，派生限制 XML 模式数据类型有效地用作 XQuery 强制类型转换的目标数据类型。

- 要转换为字符串类型的 XML 值必须满足那些 DB2 类型的长度限制，不能截断任何字符或字节。派生的 XML 模式类型的名称是大写的 SQL 类型名称加上下划线字符和字符串的最大长度；例如，如果 XMLCAST 目标数据类型是 VARCHAR(20)，那么派生的 XML 模式类型的名称是 VARCHAR_20。
- 要转换为 DECIMAL 值的 XML 值必须满足指定的 DECIMAL 值的精度，并且小数点后面的非零位数不能超过小数位数。派生的 XML 模式类型的名称是 DECIMAL_*precision_scale*，其中 *precision* 是目标 SQL 数据类型的精度，而 *scale*

是目标 SQL 数据类型的小数位；例如，如果 XMLCAST 目标数据类型为 DECIMAL (9,2)，那么派生的 XML 模式类型的名称为 DECIMAL_9_2。

- 要转换为 TIME 值的 XML 值不能在小数点后面包含具有非零位数的秒部分。派生的 XML 模式类型的名称为 TIME。

如果 XML 值不满足任何这些限制，那么派生的 XML 模式类型名将只出现在消息中。此类型名可帮助用户了解错误消息，但是不对应于已定义的任何 XQuery 类型。如果输入值不符合派生的 XML 模式类型（相应的 XQuery 目标类型）的基本类型，那么错误消息可能会指示该类型。由于此派生 XML 模式类型名的格式将来可能会更改，因此不应将它用作编程接口。

在 XQuery 强制类型转换处理 XML 值之前，将除去序列中的任何文档节点，而被除去的文档节点的每个直接子代将成为该序列中的一项。如果文档节点具有多个直接子代节点，那么修改之后的序列比原始序列具有更多项。没有任何文档节点的 XML 值就会通过使用 XQuery fn:data 函数被原子化，并将获得的原子化序列值用于 XQuery 强制类型转换。如果原子化的序列值是一个空序列，那么强制类型转换将返回空值，并且不会执行任何进一步处理。如果原子化的序列值中有多个项，那么会返回错误（SQLSTATE 10507）。

如果用于 XMLCAST 的目标类型是 SQL 数据类型 DATE、TIME 或 TIMESTAMP，那么 XQuery 强制类型转换产生的 XML 值也会被调整为 UTC，并且会除去该值的时区部分。

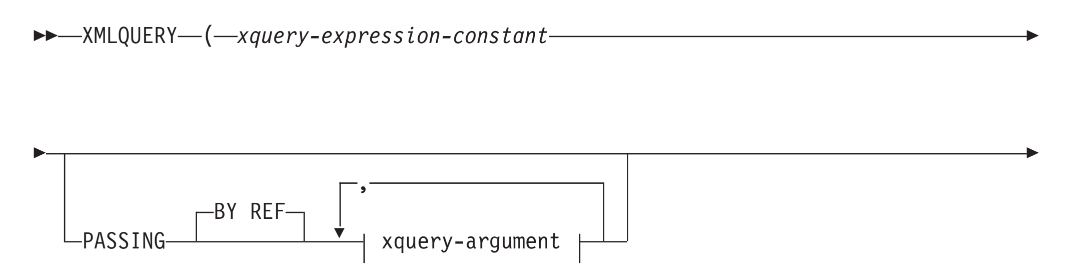
当相应的 XQuery 目标类型值被转换为 SQL 目标类型时，二进制 XML 数据类型（例如，xs:base64Binary 或 xs:hexBinary）就会从字符格式转换为实际的二进制数据。

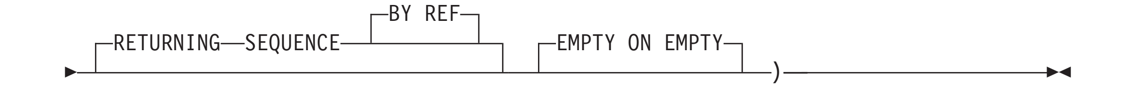
如果（通过使用 XMLCAST）将 INF、-INF 或 NaN 这些 xs:double 或 xs:float 值强制转换为 SQL 数据类型 DOUBLE 或 REAL 值，那么会返回错误（SQLSTATE 22003）。xs:double 或 xs:float 值 -0 将被转换为 +0。

如果源操作数不是用户定义的单值类型，那么目标类型可以是用户定义的单值类型。在此情况下，将使用 XMLCAST 规范将源值强制转换为用户定义的单值类型（即，目标类型）的源类型，然后，将使用 CAST 规范将此值强制转换为用户定义的单值类型。

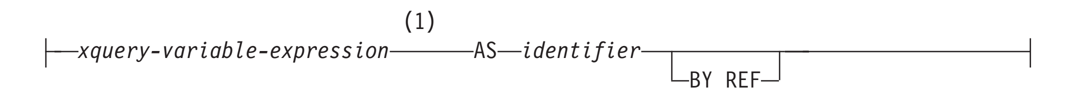
在非 Unicode 数据库中，从 XML 值强制转换为非 XML 目标类型涉及将代码页从内部 UTF-8 格式转换为数据库代码页。如果 XML 值中的任何代码点不出现在数据库代码页中，那么这种转换可能会导致引入替换字符。

XMLQUERY





xquery-argument:



注:

1 表达式的数据类型不能是 DECFLOAT。

模式为 SYSIBM。不能将函数名指定为限定名。

XMLQUERY 函数返回一个 XML 值，该值是可能使用指定输入自变量作为 XQuery 变量对 XQuery 表达式进行求值的结果。

xquery-expression-constant
指定一个 SQL 字符串常量，使用受支持的 XQuery 语言语法可将其解释为 XQuery 表达式。在解析为 XQuery 语句之前，该常量字符串被转换为 UTF-8。XQuery 表达式使用一组可选的输入 XML 值执行，并返回同样作为 XMLQUERY 表达式的值返回的输出序列。*xquery-expression-constant* 的值不能是空字符串或空白字符串（SQLSTATE 10505）。

PASSING
指定输入值以及这些值传递至 *xquery-expression-constant* 所指定的 XQuery 表达式的方式。缺省情况下，通过使用列名作为变量名，将调用函数时所在作用域中的每个唯一列名隐式传递至 XQuery 表达式。如果指定的 xquery-argument 中的 *identifier* 与列名作用域相匹配，那么会将显式 xquery-argument 传递至 XQuery 表达式，从而覆盖该隐式列。

BY REF
指定缺省传递机制供数据类型为 XML 的任何 *xquery-variable-expression* 和返回的值引用。当通过引用传递 XML 值时，XQuery 求值将直接从指定的输入表达式中使用输入节点树（如果有），这将保留所有属性，包括原始节点标识和文档顺序。如果两个参数传递相同的 XML 值，并且节点标识比较和文档顺序比较涉及这两个输入参数之间包含的某些节点，那么这两种比较可能引用相同 XML 节点树中的节点。

此子句不影响非 XML 值的传递方式。非 XML 值在强制转换为 XML 期间将创建值的新副本。

xquery-argument
指定将传递至 *xquery-expression-constant* 所指定的 XQuery 表达式的参数。参数指定值和传递该值的方式。该参数包括将进行求值的 SQL 表达式。

- 如果生成值的类型是 XML，那么它将变成 *input-xml-value*。空 XML 值将转换为 XML 空序列。
- 如果生成值的类型不是 XML，那么它必须可强制转换为 XML 数据类型。空值将转换为 XML 空序列。转换后的值将变成 *input-xml-value*。

对 *xquery-expression-constant* 进行求值时，将为 XQuery 变量提供一个等于 *input-xml-value* 的值和由 AS 子句指定的名称。

xquery-variable-expression

指定一个 SQL 表达式，其值在执行期间可供 *xquery-expression-constant* 所指定的 XQuery 表达式使用。该表达式不能包含序列引用（SQLSTATE 428F9）或 OLAP 函数（SQLSTATE 42903）。该表达式的数据类型不能是 DECFLOAT。

AS *identifier*

指定 *xquery-variable-expression* 所生成的值将作为 XQuery 变量传递至 *xquery-expression-constant*。该变量名将为 *identifier*。XQuery 语言中的变量名前面的前导美元符号（$）不包括在 *identifier* 中。标识必须是有效的 XQuery 变量名，并且限于 XML NCName （SQLSTATE 42634）。标识的长度不能超过 128 个字节。同一 PASSING 子句中的两个参数不能使用相同的标识（SQLSTATE 42711）。

BY REF

指示将通过引用传递 XML 输入值。当通过引用传递 XML 值时，XQuery 求值将直接从指定的输入表达式中使用输入节点树（如果有），这将保留所有属性，包括原始节点标识和文档顺序。如果两个参数传递相同的 XML 值，并且节点标识比较和文档顺序比较涉及这两个输入参数之间包含的某些节点，那么这两种比较可能引用相同 XML 节点树中的节点。如果在 *xquery-variable-expression* 后面未指定 BY REF，那么将使用 PASSING 关键字后面的语法所提供的缺省传递机制传递 XML 参数。不能对非 XML 值指定此选项。传递非 XML 值时，该值将转换为 XML；此过程会创建副本。

RETURNING SEQUENCE

指示 XMLQUERY 表达式返回一个序列。

BY REF

指示通过引用返回 XQuery 表达式的结果。如果此值包含节点，那么使用 XQuery 表达式的返回值的任何表达式将直接接收节点引用，这将保留所有节点属性，包括原始节点标识和文档顺序。引用的节点在其节点树内保持相连。如果未指定 BY REF 子句，但指定了 PASSING，那么将使用缺省传递机制。如果既未指定 BY REF 也未指定 PASSING，那么缺省传递机制为 BY REF。

EMPTY ON EMPTY

指定处理 XQuery 表达式所产生的空序列将作为空序列返回。

结果的数据类型为 XML；它不能为空。

如果对 XQuery 表达式进行求值时产生错误，那么 XMLQUERY 函数将返回 XQuery 错误（SQLSTATE 级别“10”）。

注:

1. **XMLQUERY 用法限制**: XMLQUERY 函数不能是:
   - 与 JOIN 运算符相关联的 ON 子句或 MERGE 语句的一部分（SQLSTATE 42972）
   - CREATE INDEX EXTENSION 语句中的 GENERATE KEY USING 或 RANGE THROUGH 子句的一部分（SQLSTATE 428E3）

- CREATE FUNCTION（外部标量）语句中的 FILTER USING 子句或 CREATE INDEX EXTENSION 语句中的 FILTER USING 子句的一部分（SQLSTATE 428E4）
- 检查约束或列生成表达式的一部分（SQLSTATE 42621）
- group-by 子句的一部分（SQLSTATE 42822）
- 列函数自变量的一部分（SQLSTATE 42607）

2. **作为子查询的 XMLQUERY**: 充当子查询的 XMLQUERY 表达式可能受限制子查询的语句限制。

XMLTABLE 函数概述

XMLTABLE 是一个 SQL 表函数，它从 XQuery 表达式的求值结果中返回表。XQuery 表达式通常返回值作为序列，但是，XMLTABLE 允许您执行 XQuery 表达式并返回值作为表。返回的表可以包含任何 SQL 数据类型（包括 XML）的列。

与 XMLQUERY 函数一样，可以将变量传递至在 XMLTABLE 中指定的 XQuery 表达式。XQuery 表达式的结果用于产生生成的表中的列值。生成的表的结构由 XMLTABLE 的 COLUMNS 子句定义。在此子句中，通过指定列名、数据类型和生成列值的方式来定义列的特征。还提供了用于传递列名而不必显式指定名称的较简单语法。请参阅第 97 页的『使用 XMLEXISTS、XMLQUERY 或 XMLTABLE 传递的简单列名』。

可以通过在 XMLTABLE 的 PATH 子句中指定 XQuery 表达式来产生生成的表中的列值。如果没有为 PATH 子句指定 XQuery 表达式，那么将列名用作 XQuery 表达式来生成列值，并且在生成列值时，先前在 XMLTABLE 中指定的 XQuery 表达式的结果将成为外部上下文项。对于生成列值的 PATH 子句的 XQuery 表达式返回空序列的情况，还可以指定可选缺省子句来为列提供缺省值。

如果生成的表中的列类型不是 XML，并且定义列值的 XQuery 表达式的结果不是空序列，那么隐式使用 XMLCAST 来将 XML 值转换为目标数据类型的值。

XMLTABLE 函数允许您选择是否声明名称空间。如果使用 XMLNAMESPACES 声明指定名称空间，那么这些名称空间绑定适用于 XMLTABLE 函数调用中的所有 XQuery 表达式。如果不使用 XMLNAMESPACES 声明来声明名称空间绑定，那么绑定仅适用于行 XQuery 表达式，这遵循名称空间声明。

XMLTABLE 优点

返回表（而不是序列）使得能够在 SQL 查询上下文中执行类似以下的操作:

- 在 SQL 全查询中迭代 XQuery 表达式的结果

  例如，在以下查询中，SQL 全查询迭代通过在 XMLTABLE 中执行 XQuery 表达式“db2-fn:xmlcolumn(″CUSTOMER.INFO″)/customerinfo”所生成的表。

  ```
  SELECT X.*
  FROM XMLTABLE ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
                COLUMNS "CUSTNAME" CHAR(30) PATH 'name',
                        "PHONENUM" XML PATH 'phone')
       as X
  ```

- 将存储的 XML 文档中的值插入到表中（请参阅有关插入值的 XMLTABLE 示例）
- 对 XML 文档中的值进行排序

例如，在以下查询中，按存储在 CUSTOMER 表的 INFO 列中的 XML 文档内的客户名对结果进行排序。

```
SELECT X.*
FROM XMLTABLE ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
               COLUMNS "CUSTNAME" CHAR(30) PATH 'name',
                       "PHONENUM" XML PATH 'phone')
     as X
ORDER BY X.CUSTNAME
```

- 将一些 XML 值存储为关系数据，并将一些 XML 值存储为 XML（请参阅有关插入值的 XMLTABLE 示例）

重要事项: 如果在 XMLTABLE 的 PATH 选项中指定的 XQuery 表达式返回:

- 多个项组成的序列，则列的数据类型必须为 XML。如果正在将从 XMLTABLE 返回的值插入到 XML 列中，那么确保插入的值是格式良好的 XML 文档。有关处理返回多个项的序列的示例，请参阅有关插入值的 XMLTABLE 示例。
- 空序列，则对该列值返回空值。

示例: 插入 XMLTABLE 中返回的值

XMLTABLE SQL 表函数可用于从存储的 XML 文档中检索值，然后将检索到的值插入到表中。

此方法是简单形式的分解，分解是将 XML 文档片段存储在关系表列中的过程。（随带注释的 XML 模式分解功能提供的是更普遍的一种分解类型。借助带注释的 XML 模式分解，可以同时将多个 XML 文档分解成多个表。）

例如，如果下列两个 XML 文档存储在名为 CUSTOMER 的表中:

```
<customerinfo Cid="1001">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
    <name>Robert Shoemaker</name>
    <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

并且您想要将这些文档中的值插入到按如下定义的表中:

```
CREATE TABLE CUSTPHONE (custname char(30), numbers XML)
```

然后，使用 XMLTABLE 的以下 INSERT 语句用 XML 文档中的值填充 CUSTPHONE:

```
INSERT INTO CUSTPHONE
    SELECT X.*
  FROM XMLTABLE ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
                COLUMNS
                  "CUSTNAME" CHAR(30) PATH 'name',
                  "PHONENUM" XML PATH 'document{<allphones>{phone}</allphones>}'
                ) as X
```

请注意，XQuery 节点构造函数“document{<allphones>{phone}</allphones>}”是在 XMLTABLE 中 PHONENUM 列的路径表达式中指定的。需要文档构造函数，因为插入到 XML 列（在此示例中为 NUMBERS 列）中的值必须是格式良好的 XML 文档。在此示例中，<customerinfo> 文档中 Cid="1003" 的所有 <phone> 元素返回在一个包含四个项的序列中:

```
{<phone type="work">905-555-7258</phone>,<phone type="home">416-555-2937</phone>,
<phone type="cell">905-555-8743</phone>, <phone type="cottage">613-555-3278</phone>}
```

此序列本身是结构不当的 XML 文档，因此不能插入到 XML 列 NUMBERS 中。要确保成功插入 phone 值，将序列的所有项构造成一个格式良好的文档。

结果表如下所示（已对该输出进行格式编排以便更加清楚明了）:

表 15. 结果表

CUSTNAME	NUMBER
Kathy Smith	<allphones> <phone type="work">905-555-7258</phone> </allphones>
Robert Shoemaker	<allphones> <phone type="work">905-555-7258</phone> <phone type="home">416-555-2937</phone> <phone type="cell">905-555-8743</phone> <phone type="cottage">613-555-3278</phone> </allphones>

示例: 使用 XMLTABLE 对某项的每个实例返回一行

如果 XML 文档包含某个元素的多个实例，并且您想要对此元素的每个实例生成一行，那么可以使用 XMLTABLE 来达到此效果。

例如，如果下列两个 XML 文档存储在名为 CUSTOMER 的表中:

```
<customerinfo Cid="1001">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
    <name>Robert Shoemaker</name>
    <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
```

```
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

要创建一个表，其中每个 <phone> 值都存储在单独行中，可以按如下所示使用 XMLTABLE:

```
  SELECT X.*
FROM CUSTOMER C, XMLTABLE ('$cust/customerinfo/phone' PASSING C.INFO as "cust"
                            COLUMNS "CUSTNAME" CHAR(30) PATH '../name',
                            "PHONETYPE" CHAR(30) PATH '@type',
                            "PHONENUM" CHAR(15) PATH '.'
                           ) as X
```

此查询对两个 XML 文档产生以下结果:

表 16. 结果表

CUSTNAME	PHONETYPE	PHONENUM
Kathy Smith	work	905-555-7258
Robert Shoemaker	work	905-555-7258
Robert Shoemaker	home	416-555-2937
Robert Shoemaker	cell	905-555-8743
Robert Shoemaker	cottage	613-555-3278

请注意名称为“Robert Shoemaker”的 XML 文档中的每个 <phone> 元素是如何在单独行中返回的。

对于相同文档，您也可以抽取 <phone> 元素作为 XML，如下所示:

```
  SELECT X.*
FROM CUSTOMER C, XMLTABLE ('$cust/customerinfo/phone' PASSING C.INFO as "cust"
                            COLUMNS "CUSTNAME" CHAR(30) PATH '../name',
                            "PHONETYPE" CHAR(30) PATH '@type',
                            "PHONENUM" XML PATH '.'
                           ) as X
```

此查询对两个 XML 文档产生以下结果（已对该输出进行格式编排以便更加清楚明了）:

表 17. 结果表

CUSTNAME	PHONETYPE	PHONENUM
Kathy Smith	work	<phone type=″work″>416-555-1358</phone>
Robert Shoemaker	work	<phone type=″work″>905-555-7258</phone>
Robert Shoemaker	home	<phone ″ type=″work″>416-555-2937</phone>
Robert Shoemaker	cell	<phone type=″work″>905-555-8743</phone>
Robert Shoemaker	cottage	<phone type=″work″>613-555-3278</phone>

示例: 使用 XMLTABLE 处理 XML 文档中多个树所包含的元素

在包含多个层次结构或树的 XML 文档中，一个树中的元素可能与另一个树中的元素之间存在关系。可以使用 XPath 表达式来处理 XML 文档中的相关元素。

以下示例将根据 XML 文档中的信息来生成有关将设备和家具搬迁至新大厦的表。此 XML 文档中包含与新大厦中各项所在位置相关的信息。

以下语句将创建用于存储此 XML 文档的 MOVE 表:

```
CREATE TABLE MOVE (ID BIGINT NOT NULL PRIMARY KEY, MOVEINFO XML )
```

以下 XML 数据包含有关各个公司拥有的各项及其位置的信息。XML 信息分成两个树。一个树中包含有关各项的信息，例如，项所属的部门、项的标记号以及对项的描述。另一个树中包含有关搬迁操作的信息，例如，部门搬迁到的新位置、为办公室指定的楼层、公共区域以及存储区。

以下语句会将此 XML 文档插入 MOVE 表中:

```
INSERT into MOVE (ID, MOVEINFO) values ( 1, '
<listing>
  <items>
   <item dept="acct" tag="12223">
       <name>laser printer</name>
       <area>common</area>
  </item>
   <item dept="recptn" tag="23665">
       <name>monitor, CRT</name>
       <area>storage</area>
  </item>
   <item dept="acct" tag="42345">
       <name>CPU, desktop</name>
       <area>office</area>
  </item>
   <item dept="recptn" tag="33301">
       <name>monitor, LCD</name>
       <area>office</area>
  </item>
   <item dept="mfg" tag="10002">
       <name>cabinet, 3 dwr</name>
       <area>office</area>
  </item>
   <item dept="acct" tag="65436">
       <name>table, round 1m</name>
       <area>storage</area>
  </item>
  </items>

  <locations>
   <building dept="recptn" >
         <wing>main</wing>
         <floor area="storage">1</floor>
         <floor area="common">1</floor>
         <floor area="office">2</floor>
   </building>

   <building dept="mfg" >
         <wing>east</wing>
         <floor area="storage">1</floor>
         <floor area="common">2</floor>
         <floor area="office">2</floor>
   </building>

   <building dept="acct" >
         <wing>west</wing>
         <floor area="storage">2</floor>
         <floor area="common">1</floor>
         <floor area="office">2</floor>
   </building>
```

```
  </locations>

</listing>
')
```

以下 SELECT 语句将项目信息与位置信息组合到一起，并创建表来列示项目信息、部门信息和新位置。

映射的关键是使用相对 XPath 轴 $x/../../ 使项目信息与位置信息匹配。

```
SELECT T.*
   from MOVE,
  XMLTABLE( '$doc/listing/items/item' PASSING MOVE.MOVEINFO AS "doc"
  COLUMNS
    ITEM_ID  VARCHAR(10) PATH 'let $x := . return $x/@tag' ,
    DESC     VARCHAR(20) PATH 'let $x := . return $x/name' ,
    DEPT     VARCHAR(15) PATH 'let $x := . return $x/@dept' ,
    WING     VARCHAR(10) PATH 'let $x := . return $x/../../locations/
               building[@dept = $x/@dept]/wing',
    FLOOR    VARCHAR(10) PATH 'let $x := . return $x/../../locations/
               building/floor[@area = $x/area
                               and ../@dept = $x/@dept ]'
  )as T
```

对样本数据运行此语句时，将返回以下数据：

```
ITEM_ID    DESC                 DEPT            WING       FLOOR
---------- -------------------- --------------- ---------- ----------
12223      printer, laser       acct            west       1
23665      monitor, CRT         recptn          main       1
42345      CPU, desktop         acct            west       2
33301      monitor, LCD         recptn          main       2
10002      cabinet, 3 dwr       mfg             east       2
65436      table, round 1m      acct            west       2
```

示例：使用 XMLTABLE 处理分层数据

XML 文档中可以包含数据的层次结构，而此层次结构包含的嵌套级别数可变。可以使用 XPath 表达式来处理分层数据。

以下示例将创建由一台计算机的部件以及每个部件的父组件组成的列表。信息基于一个 XML 文档，此文档中包含计算机的组件以及组件之间的关系。

以下语句将创建用于存储此 XML 文档的 BOMLIST 表：

```
CREATE TABLE BOMLIST (Cid BIGINT NOT NULL PRIMARY KEY, ITEMS XML )
```

XML 文档包含组件和子组件的列表。组件列表与用于描述组件的子组件的层次结构相关。如果一个组件由多个子组件组成，那么每个子组件将列示为此组件的一个子元素。

以下语句会将此 XML 文档插入 BOMLIST 表中：

```
CREATE TABLE BOMLIST (Cid BIGINT NOT NULL PRIMARY KEY, ITEMS XML )
insert into BOMLIST (Cid, ITEMS) values ( 1, '
<item desc="computersystem" model="L1234123">
   <part desc="computer" partnum="5423452345">
      <part desc="motherboard" partnum="5423452345">
         <part desc="CPU" partnum="6109486697">
            <part desc="register" partnum="6109486697"/>
        </part>
         <part desc="memory" partnum="545454232">
            <part desc="transistor" partnum="6109486697"/>
```

```
      </part>
    </part>

    <part desc="diskdrive" partnum="6345634563456">
       <part desc="spindlemotor" partnum="191986123"/>
    </part>
    <part desc="powersupply" partnum="098765343">
       <part desc="powercord" partnum="191986123"/>
    </part>
  </part>

   <part desc="monitor" partnum="898234234">
      <part desc="cathoderaytube" partnum="191986123"/>
    </part>

   <part desc="keyboard" partnum="191986123">
      <part desc="keycaps" partnum="191986123"/>
    </part>

   <part desc="mouse" partnum="98798734">
      <part desc="mouseball" partnum="98798734"/>
    </part>
</item>
')
```

以下 SELECT 语句浏览文档并创建表以列示部件和父部件。

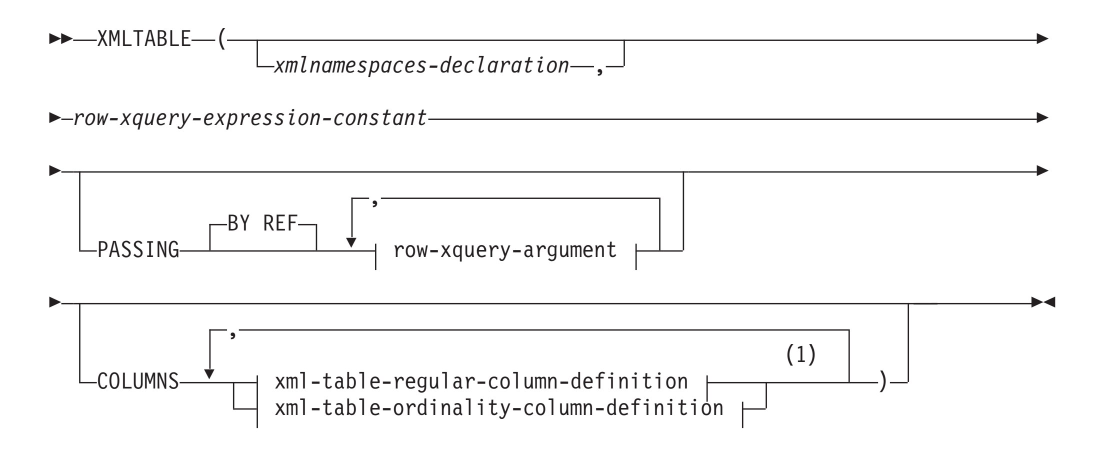
一项重要功能是使用 XMLTABLE 函数创建表 B。在 XPath 轴 `$doc//part` 中使用 `//` 来浏览“项”节点中的所有部件元素。

```
SELECT
  A.ITEMNAME,
  B.PART,
  B.PARENT
   FROM BOMLIST ,
   XMLTABLE('$doc/item' PASSING BOMLIST.ITEMS AS "doc"
    COLUMNS
      ITEMNAME VARCHAR(20) PATH './@desc',
      ITEM   XML           PATH '.'
   )AS A,
   XMLTABLE('$doc//part' PASSING A.ITEM AS "doc"
    COLUMNS
     PART  VARCHAR(20) PATH './@desc',
     PARENT VARCHAR(20) PATH '../@desc'
   )AS B
```

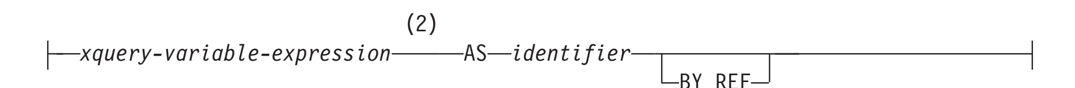
对数据运行以下语句时，将显示下表并列示部件和父部件:

```
ITEMNAME             PART                 PARENT
-------------------- -------------------- --------------------
computersystem       computer             computersystem
computersystem       motherboard          computer
computersystem       CPU                  motherboard
computersystem       register             CPU
computersystem       memory               motherboard
computersystem       transistor           memory
computersystem       diskdrive            computer
computersystem       spindlemotor         diskdrive
computersystem       powersupply          computer
computersystem       powercord            powersupply
computersystem       monitor              computersystem
computersystem       cathoderaytube       monitor
computersystem       keyboard             computersystem
computersystem       keycaps              keyboard
computersystem       mouse                computersystem
computersystem       mouseball            mouse
```

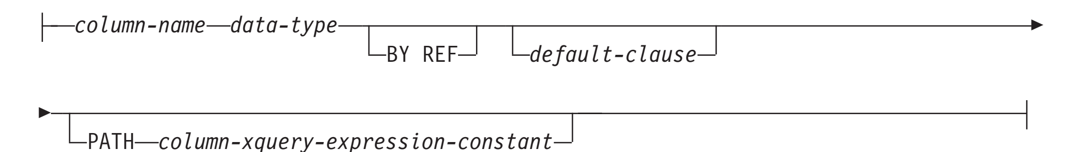
XMLTABLE



row-xquery-argument:



xml-table-regular-column-definition:



xml-table-ordinality-column-definition:

```
├─column-name─FOR ORDINALITY──────────────────────────────────────┤
```

注:

1 不能多次指定 xml-table-ordinality-column-definition 子句（SQLSTATE 42614）。

2 表达式的数据类型不能是 DECFLOAT。

模式为 SYSIBM。不能将函数名指定为限定名。

XMLTABLE 函数返回一个结果表，该结果表是可能使用指定输入自变量作为 XQuery 变量对 XQuery 表达式进行求值的结果。行 XQuery 表达式的结果序列中的每个序列项都表示结果表中的一行。

xmlnamespaces-declaration
: 指定一个或多个 XML 名称空间声明，这些声明将成为 *row-xquery-expression-constant* 和 *column-xquery-expression-constant* 的静态上下文的一部分。如果 XQuery 表达式是 XMLTABLE 的参数，那么这种表达式的静态已知名称空间集由预先建立的静态已知名称空间集和此子句中指定的名称空间声明组成。XQuery 表达式中的 XQuery 序言可以覆盖这些名称空间。

如果未指定 *xmlnamespaces-declaration*，那么只有预先建立的静态已知名称空间集适用于 XQuery 表达式。

row-xquery-expression-constant

指定一个 SQL 字符串常量，使用受支持的 XQuery 语言语法可将其解释为 XQuery 表达式。该常量字符串将直接转换为 UTF-8，而不转换为数据库或节代码页。XQuery 表达式使用一组可选的输入 XML 值执行，并返回一个输出 XQuery 序列，且在该序列中对每个序列项生成一行。*row-xquery-expression-constant* 的值不能是空字符串或空白字符串（SQLSTATE 10505）。

PASSING

指定输入值以及这些值传递至 *row-xquery-expression-constant* 所指定的 XQuery 表达式的方式。缺省情况下，通过使用列名作为变量名，将调用函数时所在作用域中的每个唯一列名隐式传递至 XQuery 表达式。如果指定的 row-xquery-argument 中的 *identifier* 与列名作用域相匹配，那么会将显式 row-xquery-argument 传递至 XQuery 表达式，从而覆盖该隐式列。

BY REF

指定缺省情况下通过引用传递任何 XML 输入参数。当通过引用传递 XML 值时，XQuery 求值将直接从指定的输入表达式中使用输入节点树（如果有），这将保留所有属性，包括原始节点标识和文档顺序。如果两个参数传递相同的 XML 值，并且节点标识比较和文档顺序比较涉及这两个输入参数之间包含的某些节点，那么这两种比较可能引用相同 XML 节点树中的节点。

此子句不影响非 XML 值的传递方式。非 XML 值在强制转换为 XML 期间将创建值的新副本。

row-xquery-argument

指定将传递至 *row-xquery-expression-constant* 所指定的 XQuery 表达式的参数。参数指定值和传递该值的方式。该参数包括在将结果传递至 XQuery 表达式之前进行求值的 SQL 表达式。

- 如果生成值的类型是 XML，那么它将变成 *input-xml-value*。空 XML 值将转换为 XML 空序列。
- 如果生成值的类型不是 XML，那么它必须可强制转换为 XML 数据类型。空值将转换为 XML 空序列。转换后的值将变成 *input-xml-value*。

对 *row-xquery-expression-constant* 进行求值时，将为 XQuery 变量提供一个等于 *input-xml-value* 的值和由 AS 子句指定的名称。

xquery-variable-expression

指定一个 SQL 表达式，其值在执行期间可供 *row-xquery-expression-constant* 所指定的 XQuery 表达式使用。该表达式不能包含 NEXT VALUE 表达式、PREVIOUS VALUE 表达式（SQLSTATE 428F9）或 OLAP 函数（SQLSTATE 42903）。该表达式的数据类型不能是 DECFLOAT。

AS *identifier*

指定 *xquery-variable-expression* 所生成的值将作为 XQuery 变量传递至 *row-xquery-expression-constant*。该变量名将为 *identifier*。XQuery 语言中的变量名前面的前导美元符号（$）不包括在 *identifier* 中。标识必须是有效的 XQuery 变量名，并且限于 XML NCName。标识的长度不能超过 128 个字节。同一 PASSING 子句中的两个参数不能使用相同的标识（SQLSTATE 42711）。

BY REF

指示将通过引用传递 XML 输入值。当通过引用传递 XML 值时，XQuery 求值将直接从指定的输入表达式中使用输入节点树（如果有），这将保留所有属性，包括原始节点标识和文档顺序。如果两个参数传递相同的 XML 值，并且节点标识比较和文档顺序比较涉及这两个输入参数之间包含的某些节点，那么这两种比较可能引用相同 XML 节点树中的节点。如果在 *xquery-expression-variable* 后面未指定 BY REF，那么将使用 PASSING 关键字后面的语法所提供的缺省传递机制传递 XML 参数。不能对非 XML 值指定此选项（SQLSTATE 42636）。传递非 XML 值时，该值将转换为 XML；此过程会创建副本。

COLUMNS

指定结果表的输出列. 如果未指定此子句，那么将通过引用返回单个未命名的 XML 数据类型列，其值基于对 *row-xquery-expression-constant* 中的 XQuery 表达式进行求值生成的序列项（相当于指定 PATH '.'）。要引用结果列，必须在该函数后面的 *correlation-clause* 中指定 *column-name*。

xml-table-regular-column-definition

指定结果表的输出列，包括列名、数据类型、XML 传递机制以及用于从表示行的序列项中抽取值的 XQuery 表达式。

column-name

指定结果表中的列名。该名称不能被限定，并且不能对表中的多列使用相同的名称（SQLSTATE 42711）。

data-type

指定列的数据类型。请参阅 CREATE TABLE 以了解语法和可用类型的描述。如果存在支持从 XML 数据类型强制转换为所指定的 *data-type* 的 XMLCAST，那么可以在 XMLTable 中使用 *data-type*。

BY REF

指定通过引用返回 XML 数据类型列的 XML 值。缺省情况下，通过引用返回 XML 值。通过引用返回 XML 值时，XML 值将直接在结果值中包括输入节点树（如果有）并保留所有属性，包括原始节点标识和文档顺序。不能对非 XML 列指定此选项（SQLSTATE 42636）。处理非 XML 列时，将转换 XML 值；此过程会创建副本。

default-clause

指定列的缺省值。请参阅 CREATE TABLE 以了解语法和 *default-clause* 的描述。对于 XMLTABLE 结果列，如果处理 *column-xquery-expression-constant* 中包含的 XQuery 表达式时返回空序列，那么将应用缺省值。

PATH *column-xquery-expression-constant*

指定一个 SQL 字符串常量，使用受支持的 XQuery 语言语法可将其解释为 XQuery 表达式。该常量字符串将直接转换为 UTF-8，而不转换为数据库或节代码页。*column-xquery-expression-constant* 指定一个 XQuery 表达式，该表达式确定与 *row-xquery-expression-constant* 中的 XQuery 表达式的求值结果项相关的列值。假定处理 *row-xquery-expression-constant* 生成的结果中的某项是外部提供的上下文项，则将对 *column-xquery-expression-constant* 进行求值并返回输出序列。按如下所示根据此输出序列确定列值:

- 如果输出序列不包含任何项，那么 *default-clause* 将提供列值。
- 如果返回空序列并且未指定 *default-clause*，那么将对列指定空值。

- 如果返回非空序列，那么将使用 XMLCAST 规范将值强制转换为对列指定的 *data-type*。在处理此 XMLCAST 时可能会返回错误。

column-xquery-expression-constant 的值不能是空字符串或空白字符串（SQLSTATE 10505）。如果未指定此子句，那么缺省 XQuery 表达式仅为 *column-name*。

xml-table-ordinality-column-definition
指定结果表的序数列。

column-name
指定结果表中的列名。该名称不能被限定，并且不能对表中的多列使用相同的名称（SQLSTATE 42711）。

FOR ORDINALITY
指定 *column-name* 是结果表的序数列。此列的数据类型为 BIGINT。结果表中此列的值是对 *row-xquery-expression-constant* 中的 XQuery 表达式进行求值生成的序列中行项的序列号。

如果对任何 XQuery 表达式进行求值时产生错误，那么 XMLTABLE 函数将返回 XQuery 错误（SQLSTATE 级别“10”）。

示例:

- 将订单中状态为“NEW”的订单项列示为表结果。

```
SELECT U."PO ID", U."Part #", U."Product Name",
     U."Quantity", U."Price", U."Order Date"
  FROM PURCHASEORDER P,
     XMLTABLE(XMLNAMESPACES('http://podemo.org' AS "pod"),
              '$po/PurchaseOrder/itemlist/item' PASSING P.PORDER AS "po"
              COLUMNS "PO ID"        INTEGER       PATH '../../@POid',
                      "Part #"       CHAR(6)       PATH 'product/@pid',
                      "Product Name" CHAR(50)      PATH 'product/pod:name',
                      "Quantity"     INTEGER       PATH 'quantity',
                      "Price"        DECIMAL(9,2)  PATH 'product/pod:price',
                      "Order Date"   TIMESTAMP     PATH '../../dateTime'
              ) AS U
   WHERE P.STATUS = 'NEW'
```

查询 XML 数据时的 XMLEXISTS 谓词

XMLEXISTS 谓词确定 XQuery 表达式是否返回一个或多个项的序列。如果在此谓词中指定的 XQuery 表达式返回空序列，那么 XMLEXISTS 返回 false；否则，返回 true。

可以在 SELECT 语句的 WHERE 子句中使用 XMLEXISTS 谓词。这种用法表示可以使用存储的 XML 文档中的值来限制 SELECT 查询作用于的行集。

例如，以下 SQL 查询说明如何使用 XMLEXISTS 谓词来将返回的行限制为仅包含 XML 文档且 <city> 元素的值为“Toronto”的那些行。（请注意，XQuery 表达式区分大小写，而 SQL 不区分大小写。）

```
SELECT Cid
FROM CUSTOMER
WHERE XMLEXISTS ('$d//addr[city="Toronto"]' passing INFO as "d")
```

请注意，在 XMLEXISTS 的 XQuery 表达式中将值传递给 XQuery 变量的方式。在此示例中，XQuery 变量 $d 绑定至 CUSTOMER 表的 INFO 列中的文档。还提供了用

于传递列名而不必在 **passing** 子句中显式指定名称的较简单语法。请参阅第 97 页的『使用 XMLEXISTS、XMLQUERY 或 XMLTABLE 传递的简单列名』。

确保正确指定了 XMLEXISTS 中的 XQuery 表达式，以便返回期望的结果。例如，假定 CUSTOMER 表的 XML INFO 列中存储了多个文档，但只有一个文档包含值为 1000 的 Cid 属性（沿指定的路径）:

```
<customerinfo Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
    <phone type="work">416-555-1358</phone>
</customerinfo>
```

由于 XQuery 表达式中的细微差别，下列两个查询返回不同的结果:

```
SELECT *
FROM CUSTOMER
WHERE XMLEXISTS ('$d/customerinfo[@Cid=1000]' passing INFO as "d")

SELECT *
FROM CUSTOMER
WHERE XMLEXISTS ('$d/customerinfo/@Cid=1000' passing INFO as "d")
```

第一个查询如期望的那样返回包含上面显示的 XML 文档的行。但是，第二个查询返回 CUSTOMER 表的所有行，原因是对于指定的 XQuery 表达式，XMLEXISTS 谓词总是返回 true。第二个查询中的 XQuery 表达式返回布尔项的序列（它是非空序列），从而导致 XMLEXISTS 总是返回 true。然后，这将导致选择 CUSTOMER 表中的每一行，但这不是想要的结果。

XMLEXISTS 谓词用途

如果 XMLEXISTS 包括具有值谓词（*expression*）的 XPath 表达式，那么用方括号将该谓词括起来，以便 [*expression*] 作为结果。用方括号将值谓词括起来可确保 *expression* 的求值结果与根据语义期望的结果一致。

XMLEXISTS 谓词行为

以下场景说明了非空序列如何导致 XMLEXISTS 的求值结果为 *true*，即使该非空序列本身包括单个值 *false* 也是如此。不会进行索引匹配，并且查询返回的结果集比期望的要大很多。通过用方括号（[]）适当地将值谓词括起来可避免该问题。

考虑一个表、一个索引和两个查询:

```
CREATE TABLE mytable (id BIGINT, xmlcol XML);
   CREATE INDEX myidx ON mytable(xmlcol)
   GENERATE KEY USING XMLPATTERN '//text()' AS SQL VARCHAR(255);

SELECT xmlcol FROM mytable
   WHERE XMLEXISTS('$doc/CUSTOMER/ORDERS/ORDERKEY/text()="A512" '
   PASSING xmlcol AS "doc")

SELECT xmlcol FROM mytable
   WHERE XMLEXISTS('$doc/CUSTOMER[ORDERS/ORDERKEY/text()="A512"] '
   PASSING xmlcol AS "doc") ;
```

此行为的原因如下：XMLEXISTS 对 XQuery 表达式进行求值，并且在结果为空序列时返回 *false (for XMLEXISTS)*，结果为非空序列时返回 *true (for XMLEXISTS)*。接下来的一个步骤在查询求值中可能是非期望步骤：在第一个查询中，表达式指示"比较订单键与 A512"。该表达式的结果为值 *false* 或 *true*，这取决于订单键的实际值。因此，XMLEXISTS 函数总是看到具有单个项的返回序列，即，为 *false* 或 *true* 的项。因为具有一个项的序列是非空序列，所以 XMLEXISTS **始终**返回整个 *true (for XMLEXISTS)*，因此查询返回所有行。如果使用 XMLEXISTS 以便限定所有行，那么不能利用索引。

下面 5 个示例是非空序列，其中 3 个序列只包含 1 项：

```
(42, 3,4,78, 1966)
(true)
(abd, def)
(false)
(5)
```

任何这种非空序列都将导致 XMLEXISTS 返回值 *true (for XMLEXISTS)*，即使它遇到的非空序列本身返回（*false*）也是如此。

在第二个查询中，XMLEXISTS 内的表达式指示"返回包含订单键等于 A512 的订单的客户"。如果文档中不存在这种客户，那么结果确实将为空序列。此查询将使用索引，并且将返回期望的结果。

XMLEXISTS 谓词用途

将整个 *expression* 放在方括号中时，其含义固定为"如果符合 [*expression*]，那么返回 XML 数据"，如果 XML 数据不符合 *expression*，那么应始终返回空序列。

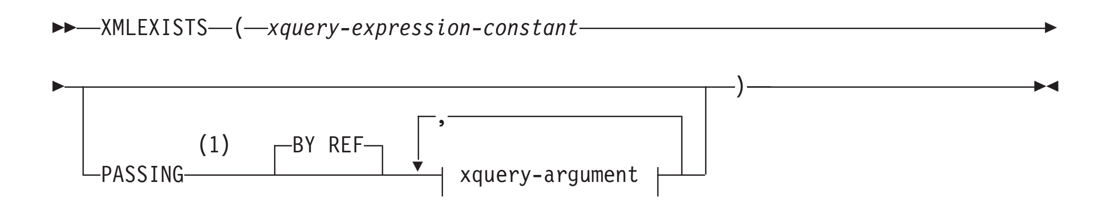
由于值比较总是在方括号内，所以下列每个 XMLEXISTS 谓词用法的样本片段都如期望的那样工作：

```
... WHERE XMLEXISTS('$doc[CUSTOMER/ORDERS/ORDERKEY/text()="A512"] '
      PASSING xmlcol as "doc") ;
   ... WHERE XMLEXISTS('$doc/CUSTOMER[ORDERS/ORDERKEY/text()="A512"] '
      PASSING xmlcol AS "doc") ;
   ... WHERE XMLEXISTS('$doc/CUSTOMER/ORDERS[ORDERKEY/text()="A512"] '
      PASSING xmlcol AS "doc") ;
```

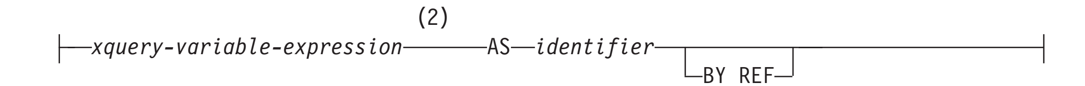
该准则还适用于没有值比较的查询，例如，想要返回正好具有 *COMMENT* 子元素的所有客户的文档：

```
... WHERE XMLEXISTS('$doc[CUSTOMER/COMMENT ] '
   PASSING xmlcol AS "doc") ;
```

XMLEXISTS 谓词



xquery-argument:



注:

1 数据类型不能是 DECFLOAT。

2 表达式的数据类型不能是 DECFLOAT。

XMLEXISTS 谓词测试 XQuery 表达式是否返回一个或多个项的序列。

xquery-expression-constant

指定一个解释为 XQuery 表达式的 SQL 字符串常量。该常量字符串将直接转换为 UTF-8，而不转换为数据库或节代码页。XQuery 表达式使用一组可选的输入 XML 值执行，并返回经过测试以确定 XMLEXISTS 谓词的结果的输出序列。*xquery-expression-constant* 的值不能是空字符串或空白字符串（SQLSTATE 10505）。

PASSING

指定输入值以及这些值传递至 *xquery-expression-constant* 所指定的 XQuery 表达式的方式。缺省情况下，通过使用列名作为变量名，将调用函数时所在作用域中的每个唯一列名隐式传递至 XQuery 表达式。如果指定的 xquery-argument 中的 *identifier* 与列名作用域相匹配，那么会将显式 xquery-argument 传递至 XQuery 表达式，从而覆盖该隐式列。

BY REF

指定缺省传递机制供数据类型为 XML 的任何 *xquery-variable-expression* 引用。当通过引用传递 XML 值时，XQuery 求值将直接从指定的输入表达式中使用输入节点树（如果有），这将保留所有属性，包括原始节点标识和文档顺序。如果两个参数传递相同的 XML 值，并且节点标识比较和文档顺序比较涉及这两个输入参数之间包含的某些节点，那么这两种比较可能引用相同 XML 节点树中的节点。

此子句不影响非 XML 值的传递方式。非 XML 值在强制转换为 XML 期间将创建值的新副本。

xquery-argument

指定将传递至 *xquery-expression-constant* 所指定的 XQuery 表达式的参数。参数指定值和传递该值的方式。该参数包括将进行求值的 SQL 表达式。

- 如果生成值的类型是 XML，那么它将变成 *input-xml-value*。空 XML 值将转换为 XML 空序列。
- 如果生成值的类型不是 XML，那么它必须可强制转换为 XML 数据类型。空值将转换为 XML 空序列。转换后的值将变成 *input-xml-value*。

对 *xquery-expression-constant* 进行求值时，将为 XQuery 变量提供一个等于 *input-xml-value* 的值和由 AS 子句指定的名称。

xquery-variable-expression

指定一个 SQL 表达式，其值在执行期间可供 *xquery-expression-constant* 所指定的 XQuery 表达式使用。该表达式不能包含序列引用（SQLSTATE 428F9）或 OLAP 函数（SQLSTATE 42903）。该表达式的数据类型不能是 DECFLOAT。

AS *identifier*

指定 *xquery-variable-expression* 所生成的值将作为 XQuery 变量传递至 *xquery-expression-constant*。该变量名将为 *identifier*。XQuery 语言中的变量名前面的前导美元符号（$）不包括在 *identifier* 中。标识必须是有效的 XQuery 变量名，并且限于 XML NCName。标识的长度不能超过 128 个字节。同一 PASSING 子句中的两个参数不能使用相同的标识（SQLSTATE 42711）。

BY REF

指示将通过引用传递 XML 输入值。当通过引用传递 XML 值时，XQuery 求值将直接从指定的输入表达式中使用输入节点树（如果有），这将保留所有属性，包括原始节点标识和文档顺序。如果两个参数传递相同的 XML 值，并且节点标识比较和文档顺序比较涉及这两个输入参数之间包含的某些节点，那么这两种比较可能引用相同 XML 节点树中的节点。如果在 *xquery-variable-expression* 后面未指定 BY REF，那么将使用 PASSING 关键字后面的语法所提供的缺省传递机制传递 XML 参数。不能对非 XML 值指定此选项。传递非 XML 值时，该值将转换为 XML；此过程会创建副本。

注意

XMLEXISTS 谓词不能是:

- 与 JOIN 运算符相关联的 ON 子句或 MERGE 语句的一部分（SQLSTATE 42972）
- CREATE INDEX EXTENSION 语句中的 GENERATE KEY USING 或 RANGE THROUGH 子句的一部分（SQLSTATE 428E3）
- CREATE FUNCTION（外部标量）语句中的 FILTER USING 子句或 CREATE INDEX EXTENSION 语句中的 FILTER USING 子句的一部分（SQLSTATE 428E4）
- 检查约束或列生成表达式的一部分（SQLSTATE 42621）
- group-by 子句的一部分（SQLSTATE 42822）
- 列函数自变量的一部分（SQLSTATE 42607）

涉及子查询的 XMLEXISTS 谓词可能受限制子查询的语句限制。

示例

```
SELECT c.cid FROM customer c
  WHERE XMLEXISTS('$d/*:customerinfo/*:addr[ *:city = "Aurora" ]'
    PASSING info AS "d")
```

在 SQL 语句与 XQuery 表达式之间传递参数

发出合并的 SQL 语句和 XQuery 表达式时，可在语句与表达式之间传递数据以修改语句和表达式的执行方式。

传递至 XMLEXISTS 和 XMLQUERY 的常量和参数标记

XMLEXISTS 谓词和 XMLQUERY 标量函数在 SQL 语句中执行 XQuery 表达式。使用常量和参数标记将 SQL 语句中的数据传递至 SQL 语句中执行的 XQuery 表达式中的变量。

可以在 XMLEXISTS 和 XMLQUERY 中将 XQuery 变量指定为 XQuery 表达式的一部分。通过 passing 子句将值传递到这些变量中。这些值是 SQL 表达式。因为传递至 XQuery 表达式的值是非 XML 值，所以必须隐式或显式将它们转换为 DB2 XQuery 支持的类型。有关受支持的数据类型转换的更多信息，请参阅有关在数据类型之间转换的文档。

用于将常量和参数标记传递至 XMLQUERY 的方法与用于传递至 XMLEXISTS 的方法相同，但 XMLEXISTS 用法更常见。这是因为在 SELECT 子句中使用 XMLQUERY 中已参数化的谓词时，不除去结果集中的任何行。相反，这些谓词用于确定将返回的文档片段。要真正除去结果集中的行，应在 WHERE 子句中使用 XMLEXISTS 谓词。因此，不会返回包含空序列的行作为结果集的一部分。此处讨论的示例说明了 XMLEXISTS 的这种更常见用法。

示例: 隐式数据类型转换

在以下查询中，在 XMLEXISTS 谓词中将 SQL 字符串常量“Aurora”（它不是 XML 类型）隐式转换为 XML 类型。在隐式数据类型转换之后，该常量具有 XML 模式子类型 xs:string，并且绑定至变量 $cityName。然后，可以在 XQuery 表达式的谓词中使用此常量。

```
SELECT XMLQUERY ('$d/customerinfo/addr' passing c.INFO as "d")
FROM Customer as c
WHERE XMLEXISTS('$d//addr[city=$cityName]'
                passing c.INFO as "d",
                'Aurora' AS "cityName")
```

示例: 显式数据类型转换

在以下查询中，因为不能确定参数标记的类型，所以必须将其显式转换为数据类型。显式转换为 SQL VARCHAR 类型的参数标记随后隐式转换为 xs:string XML 模式类型。

```
SELECT XMLQUERY ('$d/customerinfo/addr' passing c.INFO as "d")
FROM Customer as c
WHERE XMLEXISTS('$d//addr[city=$cityName]'
                passing c.INFO as "d",
                CAST (? AS VARCHAR(128)) AS "cityName")
```

使用 XMLEXISTS、XMLQUERY 或 XMLTABLE 传递的简单列名

要简化 XMLEXISTS 谓词、XMLQUERY 标量函数或 XMLTABLE 表函数的用法，可在 XMLEXISTS、XMLQUERY 或 XMLTABLE 指定的 XQuery 表达式中将列名用作参数，而不在 **passing** 子句中指定名称。

如果要使用的参数名不同于要传递的列名，那么必须使用 **passing** 子句来将列名作为参数传递。

如果 **passing** 子句中显式指定了变量并且名称与 XQuery 表达式引用的任意变量发生冲突，那么会将优先权给予 **passing** 子句中的变量。假定 CUSTOMER 表包含两个 XML 列： INFO 和 CUST，以下示例将从 INFO 列中检索 XML 数据:

```
SELECT XMLQuery('$CUST/customerinfo/name' PASSING INFO as "CUST") FROM customer
```

passing 子句中指定的变量 CUST 将用于替换 XQuery 表达式中的列 INFO。将不使用 CUSTOMER 表中的列 CUST。

示例: XMLQUERY 和 XMLEXISTS

注意: 列名在这些示例中是区分大小写的，原因是它们是用双引号括起来的。如果未括在双引号中，那么常规列名规则适用: 列名是不区分大小写的，并且存储为大写形式。

以下示例显示若干 SELECT 语句，它们返回 PURCHASEORDER 表中的文档序列。XML 文档在列 PORDER 中。前两个 SELECT 语句使用 **passing** 子句将列名 PORDER 传递至 XMLQUERY 标量函数的 XQuery 表达式。第三个 SELECT 将 PORDER 列名用作隐式传递的参数:

```
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name'  PASSING porder AS "PORDER")
   FROM purchaseorder
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name' PASSING porder AS "porder")
   FROM purchaseorder
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name') FROM purchaseorder
```

以下两个示例显示同时使用 XMLQUERY 和 XMLEXISTS 的若干函数调用。两个示例都会返回 CUSTOMER 表中的文档序列。

以下示例使用 **passing** 子句以显式方式将 INFO 列名指定为 XMLQUERY 标量函数和 XMLEXISTS 谓词中的参数:

```
SELECT XMLQUERY ('$INFO/customerinfo/addr' passing Customer.INFO as "INFO")
FROM Customer
WHERE XMLEXISTS('$INFO//addr[city=$cityName]'
                 passing Customer.INFO as "INFO",
                 'Aurora' AS "cityName")
```

在以下示例中，XMLQUERY 函数未使用传递子句，并且 XMLEXISTS **passing** 子句未指定 INFO 列。列名 INFO 将隐式传递至 XMLQUERY 标量函数和 XMLEXISTS 谓词中的 XQuery 表达式:

```
SELECT XMLQUERY ('$INFO/customerinfo/addr')
FROM Customer
WHERE XMLEXISTS('$INFO//addr[city=$cityName]'
                 passing 'Aurora' AS "cityName")
```

示例: XMLTABLE

以下两个示例显示两个使用 XMLTABLE 表函数的 INSERT 语句。两个示例都会将相同数据插入到表 T1 的表 CUSTOMER 中。表 T1 包含 XML 列 CUSTLIST。XMLTABLE 函数检索列 T1.CUSTLIST 中的数据并返回带有以下三列的表: Cid、Info 和 History。INSERT 语句将 XMLTABLE 函数中的数据插入到表 CUSTOMER 的 3 列中。

以下示例使用 **passing** 子句以显式方式将 CUSTLIST 列名指定为 XMLTABLE 表函数中的参数:

```
INSERT INTO customer SELECT X.* FROM T1,
    XMLTABLE( '$custlist/customers/customerinfo' passing T1.custlist as "custlist"
    COLUMNS
    "Cid" BIGINT PATH '@Cid',
    "Info" XML PATH 'document{.}',
    "History" XML PATH 'NULL') as X
```

在以下示例中，XMLTABLE 表函数未使用 **passing** 子句。XMLTABLE 将表 T1 中的列名 CUSTLIST 用作隐式传递参数:

```
INSERT INTO customer SELECT X.* FROM T1,
    XMLTABLE( '$custlist/customers/customerinfo'
    COLUMNS
    "Cid" BIGINT PATH '@Cid',
    "Info" XML PATH 'document{.}',
    "History" XML PATH 'NULL') as X
```

将参数从 XQuery 传递至 SQL

在 XQuery 表达式中，db2-fn:sqlquery 函数将执行检索 XML 节点序列的 SQL 全查询。使用 db2-fn:sqlquery 时，请使用 PARAMETER 函数引用从 XQuery 表达式传递至由 db2-fn:sqlquery 指定的 SQL fullselect 语句的数据。

通过使用 PARAMETER 参数，可将参数指定为 db2-fn:sqlquery 中 SQL 全查询表达式的一部分。如果在 db2-fn:sqlquery 调用中使用 PARAMETER 函数，那么还必须指定将由 PARAMETER 函数使用的 XQuery 表达式。在处理 SQL 全查询期间，每个 PARAMETER 函数调用将替换为 db2-fn:sqlquery 函数调用中对应 XQuery 表达式的结果值。可在同一 SQL 语句中多次引用 PARAMETER 函数提供的值。

db2-fn:sqlquery 函数调用中的 XQuery 表达式返回一个值。因为传递至全查询的值是 XML 值，所以它们必须隐式或显式转换为 DB2 SQL 支持的类型。有关受支持的数据类型转换的更多信息，请参阅 db2-fn:sqlquery 文档和有关在数据类型之间转换的文档。

示例：将参数传递至 db2-fn:sqlquery

以下示例是使用 db2-fn:sqlquery 的 XQuery 表达式。在处理 db2-fn:sqlquery 函数期间，对 `parameter(1)` 的两次引用会返回订单日期属性 `$po/@OrderDate` 的值。

对 DB2 SAMPLE 数据库运行时，XQuery 表达式会返回推广日期内售出的所有部件的购买标识、部件标识和购买时间。

```
  xquery
for $po in db2-fn:xmlcolumn('PURCHASEORDER.PORDER')/PurchaseOrder,
  $item in $po/item/partid
for $p in db2-fn:sqlquery(
  "select description
  from product
  where promostart < parameter(1)
  and
  promoend  >  parameter(1)",
  $po/@OrderDate )
where $p//@pid = $item
return
<RESULT>
  <PoNum>{data($po/@PoNum)}</PoNum>
  <PartID>{data($item)} </PartID>
  <PoDate>{data($po/@OrderDate)}</PoDate>
</RESULT>
```

使用 XQuery 检索数据

XQuery 规范将 XQuery 表达式的结果定义为包含 0 个项、1 个项或多个项的序列。可通过将 XQuery 用作主语言，或者将 SQL 用作主语言并使用 XMLQUERY SQL 函数来执行 XQuery 表达式。使用任一种方法执行 XQuery 表达式时，将返回 XML 序列。

根据是将 SQL 还是 XQuery 用作主语言，生成的序列出现在结果集中的方式将有所不同:

XQuery 作为主语言

通过将 XQuery 用作主语言来执行 XQuery 表达式时，结果将以结果表（具有类型为 XML 的一列）的形式返回至客户机应用程序。此结果表中的每行是对 XQuery 表达式进行求值所生成的序列中的一项。应用程序使用游标从此结果表中访存时，每次访存都会检索生成的序列中序列化的项。

SQL 作为主语言，使用 XMLQUERY

XMLQUERY 是返回 XML 值的标量函数。返回的值是包含零项、一项或多项的序列。生成的序列中的所有项都以单个序列化值的形式返回至应用程序。

要从使用 XQuery 或 XMLQUERY 的查询中访存结果，就像通常访存任何其他结果集一样从应用程序中访存结果。将应用程序变量绑定至结果集并进行访存，直到到达结果集末尾。如果 XQuery 表达式（直接发出或通过 XMLQUERY 发出）返回空序列，那么结果集中的行也为空。

管理查询结果集

如果应用程序要求在使用 XQuery 查询时返回的 XML 值是格式良好的 XML 文档（例如，如果您打算将这些值插入到类型为 XML 的列中），那么通过在 XQuery 表达式中包括元素或文档构造函数可确保值表示格式良好的 XML 文档。

示例：XQuery 和 XMLQUERY 产生的结果集之间的差别

以下示例说明了这两种查询方法的结果集之间的差别。

如果下列两个 XML 文档存储在 XML 列中，要检索所有 <phone> 元素，可以使用 XQuery 或 XMLQUERY。但是，这两个方法返回的结果集不同，并且应该由应用程序在从结果集访存时进行相应处理。

```
<customerinfo Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
    <phone type="work">416-555-1358</phone>
</customerinfo>

<customerinfo Cid="1003">
    <name>Robert Shoemaker</name>
    <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
    <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

通过将 XQuery 用作主语言来执行 XQuery 表达式在结果集中产生 5 行，如下所示：

```
XQUERY
  db2-fn:xmlcolumn ('CUSTOMER.INFO')/customerinfo/phone
```

表 18. 将 XQuery 用作主语言的结果集

<phone type=″work″>416-555-1358</phone>
<phone type=″work″>905-555-2937</phone>
<phone type=″home″>416-555-2937</phone>
<phone type=″cell″>905-555-8743</phone>
<phone type=″cottage″>613-555-3278</phone>

通过 XMLQUERY 执行 XQuery 表达式在结果集中产生 2 行，如下所示，其中表的第二行中的所有 <phone> 元素并置成单个标量值（XML 序列）：

```
SELECT XMLQUERY ('$doc/customerinfo/phone' PASSING INFO AS "doc")
FROM CUSTOMER
```

表 19. 将 SQL 用作主语言的结果集

<phone type=″work″>416-555-1358</phone>
<phone type=″work″>905-555-2937</phone><phone type=″home″>416-555-2937</phone><phone type=″cell″>905-555-8743</phone><phone type=″cottage″>613-555-3278</phone>

请注意，此结果集的第二行中包含一个表示结构不当的 XML 文档的值。

由于 XMLQUERY 是标量函数，所以结果集中存在这些差别。此函数对表中的每行执行，并且从表的某一行生成的序列构成结果集的一行。但是，XQuery 返回序列中的每个项作为结果集的单个行。

示例：管理查询结果集

在此示例中，可以修改上一个 SQL 查询以包括 XQuery 文档节点构造函数，该函数确保生成的行全部包含格式良好的文档：

```
SELECT XMLQUERY ('document{<phonelist>{$doc/customerinfo/phone}</phonelist>}'
  PASSING INFO AS "doc")
FROM CUSTOMER
```

假定数据库中存在先前显示的那些文档，通过此查询生成的表如下所示。

表 20. 使用元素构造函数生成格式良好的文档

<phonelist><phone type=″work″>416-555-1358</phone></phonelist>
<phonelist><phone type=″work″>905-555-7258</phone><phone type=″home″>416-555-2937</phone><phone type=″cell″>905-555-8743</phone><phone type=″cottage″>613-555-3278</phone></phonelist>

用于匹配索引与查询的准则概述

本节提供了一些用于将查询与基于 XML 数据的索引匹配的准则和示例。

查询是否可以使用索引取决于您创建的索引是否与查询兼容（也称为索引匹配），以及优化器是否选择在查询求值期间执行索引扫描。说明设施的访问方案将告知您查询求值是否涉及索引扫描。

查询必须至少满足下列条件才能使用XML 数据索引:

- 查询搜索条件中的数据类型与已建立索引的数据类型匹配。
- 查询搜索条件包括对其建立了索引的一部分节点。

SQL 和 XQuery 优化器

优化器规划查询求值并选择查询期间要使用的索引。查询编译期间，查询与 XML 索引定义中的所有模式匹配，以查找包含实现某部分查询所需的足够信息的候选索引。

优化器在查询求值期间可能执行下列步骤之一:

- 不使用索引扫描包含 XML 文档的表
- 使用关系索引
- 使用关系索引“与”（AND）运算或索引“或”（OR）运算
- 使用新的 XML 索引运算符
- 使用XML 数据索引对单个 XML 模式进行求值
- 使用XML 数据索引“与”（AND）运算和“或”（OR）运算对单次查询中的复杂 XML 模式进行求值

说明工具

说明设施和 Visual Explain 工具为您提供用于查询求值的访问方案。查看访问方案时，下列运算符将告诉您在查询求值期间正在使用一个还是几个索引:

IXAND
: 对来自两个或多个索引扫描的行标识执行“与”（AND）运算。

XISCAN
: 扫描基于 XML 数据的索引。

XANDOR
: 允许将进行了“与”（AND）运算的谓词应用于多个 XML 索引。

索引定义的限制

对查询的求值是否会频繁使用索引取决于索引定义相对于查询的限制性。下列示例显示可一起使用的若干查询和索引。

带有范围谓词的查询的索引

以下查询使用 XML 列 *companydocs* 在表 *companyinfo* 中检索公司信息以查找薪水超过 *35000* 的职员:

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp[@salary > 35000]'
   PASSING companydocs AS "x")
```

为了保持兼容，XML 数据索引需要将职员薪水属性节点包括在已建立索引的节点中，并将值作为 DOUBLE 类型存储。

查询可以使用下列基于 XML 数据的索引之一，例如:

```
CREATE INDEX empindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '//@salary' AS SQL DOUBLE

CREATE INDEX empindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '/company/emp/@salary'
   AS SQL DOUBLE
```

可由多个查询使用的索引

以下查询检索公司信息以查找职员标识为 *31664* 职员。

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp[@id="31664"]'
   PASSING companydocs AS "x")
```

另一查询检索公司信息以找到标识为 *K55* 的部门。

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp/dept[@id="K55"]
   PASSING companydocs AS "x")
```

为了与这两个查询兼容，XML 数据索引需要将职员标识属性节点和部门标识属性节点包括在已建立索引的节点中，并将索引中的值作为 VARCHAR 类型存储。

查询可以使用以下XML 数据索引:

```
CREATE INDEX empdeptindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(25)
```

限制 XQuery 谓词时包含名称空间

考虑以下表，其 XML 列中包含客户信息，并对该 XML 列创建了索引:

```
CREATE TABLE customer(xmlcol XML) %

CREATE UNIQUE INDEX customer_id_index ON customer(xmlcol)
   GENERATE KEY USING XMLPATTERN
   'DECLARE DEFAULT ELEMENT NAMESPACE
   "http://mynamespace.org/cust";/Customer/@id'
   AS SQL DOUBLE %
```

注: 此部分中使用的语句终止符是百分号（%），因为分号（;）已充当名称空间定界符。

以下查询未能与索引匹配:

```
SELECT xmlcol FROM customer
   WHERE XMLEXISTS('$xmlcol/*:Customer[@id=1042]'
   PASSING xmlcol AS "xmlcol") %
```

为了使查询能够使用索引，查询的限制必须和索引的限制同样多，或比索引的限制更多。索引 customer_id_index 仅包括一个特定名称空间（*http://mynamespace.org/cust*）中的 customer 元素。由于查询中使用 **:* 来表示任何名称空间，因此未使用索引。如果期望 **:* 与索引定义中的名称空间匹配，那么这与期望相反。

为了使查询使用索引，索引的限制需要较少，或查询的限制需要较多。

对于相同查询，可以成功使用以下较少限制的索引 *customer_id_index2*:

```
CREATE UNIQUE INDEX customer_id_index2 ON customer(xmlcol)
      GENERATE KEY USING XMLPATTERN '/*:Customer/@id' AS SQL DOUBLE %
```

以下限制更多的查询可以使用初始索引 *customer_id_index*:

```
SELECT xmlcol FROM customer
   WHERE XMLEXISTS('
   DECLARE NAMESPACE ns = "http://mynamespace.org/cust";
   $xmlcol/ns:Customer[@id=1042]'
   PASSING xmlcol AS "xmlcol") %
```

查询中显式指定了适当的名称空间时，可以使用索引 *customer_id_index*，因为现在查询和索引的限制同样多。还可以使用索引 *customer_id_index2*，因为它的限制比此示例中查询的限制要少。

指定 text() 节点时的注意事项

使用 XML 模式表达式包括 text() 节点可能会影响索引条目的生成。应在索引定义和谓词中一致地使用 /text()。

指定 text() 节点对索引键的影响

考虑以下样本 XML 文档分段:

```
<company name="Company1">
      <emp id="31201" salary="60000" gender="Female">
     <name><first>Laura</first><last>Brown</last></name>
     <dept id="M25">
   Finance
       </dept>
      </emp>
</company>
```

如果创建以下索引并在模式末尾指定 text()，那么不会插入索引条目，因为样本 XML 文档片段中的 *name* 元素本身不包含文本。仅子元素 *first* 和 *last* 中包含文本。

```
CREATE INDEX nameindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/name/text()' AS SQL
      VARCHAR(30)
```

但是，如果创建下一个索引并在模式末尾指定元素 *name*，那么将 *first* 和 *last* 子元素中的文本并置在插入的索引条目中。

```
CREATE INDEX nameindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/name'
      AS SQL VARCHAR(30)
```

存在或缺少 text() 节点将影响非叶子元素的索引条目生成，但对叶子元素没有影响。如果要对叶子元素建立索引，那么建议您不要指定 text()。如果指定 text()，那么查询还必须使用 text() 才能成功匹配索引。此外，模式验证仅适用于元素，不适用于文本节点。

指定可以与不包含 text() 的非叶子节点的元素匹配的 XML 模式时必须小心。并置后代元素文本节点可能导致意外结果。特别是使用 XML 模式指定 *//** 很可能对非叶子元素建立索引。

在一些情况下，并置对于使用 VARCHAR 的索引很有用。例如，下面文档片段中 */title* 的索引对于忽略标题内的粗体格式很有用:

```
<title>This is a <bold>great</bold> book about XML</title>
```

可以按如下所示编写用于查找特定职员姓名的查询谓词:

```
db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[name='LauraBrown']
```

空格在谓词和文档中很重要。如果在谓词中的“Laura”和“Brown”之间插入空格，那么以下查询不会返回任何内容，因为样本 XML 文档片段本身中的名与姓之间不包含空格:

```
db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[name='Laura Brown']
```

带有复合等于谓词的查询的索引

以下查询检索公司信息以查找财务部或市场部中的职员。

```
SELECT companydocs FROM companyinfo WHERE
   XMLExists('$x/company/emp[dept/text()='Finance'
   or dept/text()='Marketing']'
   PASSING companydocs AS "x")
```

为了保持兼容，XML 数据索引需要对已建立索引的节点中每个职员所在部门的文本节点建立索引，以便将值作为 VARCHAR 类型存储。

查询可以使用以下XML 数据索引:

```
CREATE INDEX empindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '/company/emp/dept/text()'
   AS SQL VARCHAR(30)
```

字面值的数据类型

为了使查询能够使用索引，字面值的数据类型需要与索引的数据类型匹配。

匹配字面值的数据类型

以下查询检索公司信息以查找标识为 *31201* 职员。

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp[@id="31201"]'
   PASSING companydocs AS "x")
```

为了保持兼容，XML 数据索引需要将职员标识属性节点包括在已建立索引的节点中，并将索引中的值作为 VARCHAR 类型存储。

```
CREATE INDEX empindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '/company/emp/@id'
   AS SQL VARCHAR(5)
```

如果类似的索引定义了 AS SQL DOUBLE，那么查询不能使用该索引，因为查询谓词包括字符串比较。谓词 *@id="31201"* 中使用的双引号使其成为字符串比较，这只能使用字符串索引（VARCHAR）而不能使用数字索引（DOUBLE）进行求值。

要突出显示数字谓词和字符串谓词之间的差别，考虑以下不相等谓词:

```
@id > 3
@id > "3"
```

数字谓词 *@id > 3* 与字符串谓词 *@id > "3"* 不同。*@id* 值 *10* 符合数字谓词 *@id > 3*，但不符合字符串谓词 *@id > "3"*，原因是在字符串比较中 *"3"* 大于 *"10"*。

连接谓词转换

应该在两方面将连接谓词转换为适当的数据类型。

哪些连接谓词会阻止使用索引?

考虑两个表，其 XML 列中分别存储客户信息和采购订单:

```
CREATE TABLE customer(info XML);
```

```
CREATE TABLE PurchaseOrder(POrder XML);
```

包含客户信息的 XML 文档包括属性 @cid，即数字客户标识（*cid*）。包含采购订单信息的 XML 文档也包括 *@cid* 属性，因此每个订单与一个特定客户唯一关联。由于我们希望经常按 *cid* 搜索客户和订单，所以定义索引非常有意义:

```
CREATE UNIQUE INDEX idx1 ON customer(info)
   GENERATE KEY USING XMLPATTERN '/customerinfo/@cid' AS SQL DOUBLE;

CREATE INDEX idx2 ON PurchaseOrder(POrder)
   GENERATE KEY USING XMLPATTERN '/porder/@cid' AS SQL DOUBLE;
```

我们希望查找特定邮政编码内所有客户的采购订单。直觉上，可以编写类似以下的查询:

```
XQUERY
   for $i in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
   for $j in db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/porder[@cid = $i/@cid]
   where $i/zipcode = "95141"
   return $j;
```

请注意，连接谓词 @cid = $i/@cid 要求采购订单的 *cid* 等于客户的 *cid*。

此查询返回正确的结果，但不能使用这两个索引。以嵌套循环连接的形式执行查询，并对这两个表进行表扫描。为了避免重复的表扫描，首选对 *customer* 进行单次表扫描，以查找邮政编码 *95141* 内的所有客户，然后索引使用 @cid 在采购订单表中查找。请注意，必须扫描 customer 表，因为没有 *zipcode* 的索引。

未使用索引，因为这样做不正确。如果使用了索引，那么 DB2 可能错过一些匹配的采购订单并返回不完整的结果。这是因为 @cid 属性中的某些值潜在地可能是非数字。例如，@cid 可能等于 *YPS*，因此它不包括在定义了 AS SQL DOUBLE 的数字索引中。

注: 如果已建立索引的节点的值不能转换为指定的索引数据类型，那么不会插入该值的索引条目，并且不会产生错误或警告。

允许将连接谓词与索引配合使用

如果我们确定所有 @cid 值都是数字，那么可以允许期望使用的索引。如果显式转换连接谓词以与索引的类型匹配，那么将使用索引:

```
XQUERY
   for $i in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
   for $j in db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/porder
   where $i/@cid/xs:double(.) = $j/@cid/xs:double(.)
   and $i/zipcode = "95141"
   return $j;
```

直观地说，转换建议 DB2 仅应考虑可转换为 DOUBLE 的匹配 @cid 属性。在这种指示下，我们可以确定所有必需的匹配项都出现在定义了 AS SQL DOUBLE 的索引中，因此使用该索引很安全。如果有一个文档中确实存在非数字 @cid 值，那么转换将失败并且出现运行时错误。

请注意，在 XQuery 内，数据类型转换仅对单元素起作用。特别是对于一些元素（以下示例中为 *a*、*b* 和 *c*），建议您按如下所示转换它们:

```
/a/b/c/xs:double(.)
```

如果要按如下所示转换元素，那么在任何给定元素 *b* 中存在多个元素 *c* 的情况下，将产生运行时错误:

```
/a/b/xs:double(c)
```

对于定义了 AS SQL VARCHAR 的索引，相应的连接谓词需要使用 fn:string() 函数将已比较的值转换为 xs:string 数据类型。相同操作适用于 DATE 和 TIMESTAMP 索引。以下示例说明如何在字符串连接中使用 fn:string() 函数：

```
XQUERY
   for $i in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
   for $j in db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/porder
   where $i/zipcode/fn:string(.) = $j/supplier/zip/fn:string(.)
   return <pair>{$i}{$j}</pair>
```

连接谓词的转换规则总结

下表概括应如何在两端将连接谓词转换为相应的数据类型以允许使用索引。

表 21. 连接谓词的转换规则

索引 SQL 类型	将连接谓词转换为 XML 类型
DOUBLE	xs:double
VARCHAR *integer*, VARCHAR HASHED	xs:string
DATE	xs:date
TIMESTAMP	xs:dateTime

模糊查询求值

查询可能会进行不确定地求值，并且在未涉及索引扫描时返回错误。当查询求值涉及到索引扫描时，相同查询可能会返回匹配的 XML 数据并且不产生错误，原因是导致错误的不可进行数据类型转换的 XML 片段未包含在索引中。

此示例使用包含 XQuery 表达式的以下 VALUES 语句。此表达式使用数字比较来返回其标识为 17 的 XML 文档：

```
VALUES( XMLQUERY('
  for $i in db2-fn:xmlcolumn("T.DOC")
   where $i/emp/id = 17
   return $i'))
```

T 表由单个 XML 列 DOC 组成，并且包含两个 XML 文档。下列语句将创建此表并插入文档：

```
CREATE TABLE t (doc XML) ;
INSERT INTO t VALUES ( '<emp><id>17</id></emp>' );
INSERT INTO t VALUES ( '<emp><id>ABC</id></emp>' );
```

除非使用一个定义为 SQL DOUBLE 的 XML 数据索引来查找表中匹配的文档，否则此 XQuery 表达式将返回错误。以下 CREATE INDEX 语句将创建索引：

```
CREATE INDEX EMPDBL ON t (doc) GENERATE KEY USING XMLPATTERN '/emp/id'
    as SQL DOUBLE IGNORE INVALID VALUES
```

如果表中不存在索引，那么 **where** 子句中的数字比较操作将同时应用于表中的匹配文档和不匹配的文档。将比较操作应用于不匹配的文档时会导致运行时错误，原因是无法将值 ABC 转换为数字。如果表中存在对职员标识建立索引的 XML 数据索引，那么此表达式将使用此索引，并返回第一个 XML 文档且不产生错误。第二个文档中不可进行数据类型转换的值未包括在 XML 数据索引中，原因是子句 IGNORE INVALID VALUES 指定创建索引期间忽略无效模式值并且不对其建立索引。忽略无效值是缺省选项。

Explain 工具提供的访问方案指出查询求值是否涉及到索引扫描。

要确保创建索引期间所有模式值有效，请使用 REJECT INVALID VALUES 子句。创建或更新索引期间出现无效模式值时，将发生错误。

XML 文档中的全文本搜索

可通过 DB2 Net Search Extender 对本机存储的 XML 数据执行全文本搜索。

DB2 Net Search Extender

DB2 Net Search Extender 完全支持 XML 数据类型。它使得可以对存储在 XML 列中的文档建立全文本索引。通过对 XML 列创建文本索引，可以查询 XML 文档中的所有文本并执行一些搜索（如相似搜索或通配符搜索）。DB2 Net Search Extender 是用于 Linux、UNIX 和 Windows 的 DB2 数据服务器产品的一部分，但必须单独安装。

以下示例显示一个简单的全文本搜索，它在 DEPTDOC 列中存储的 XML 文档的路径 /dept/description 中的任意位置查找单词“marketing”:

```
SELECT DEPTDOC
FROM DEPT
WHERE contains (DEPTDOC, SECTIONS("/dept/description") "marketing") = 1
```

DB2 Net Search Extender 提供的 contains 函数在路径 /dept/description 中的任何文本（包括元素或属性名以及元素或属性值）中搜索字符串“marketing”。

要使用全文本搜索，必须使用 SQL。但是，仍可以将 SQL 查询的结果返回至 XQuery 上下文以进行进一步处理。以下示例显示使用全文本搜索的 SQL 查询产生的结果如何参与 XQuery 表达式:

```
XQUERY for $i in db2-fn:sqlquery ('SELECT DEPTDOC FROM DEPT
    WHERE contains
    (DEPTDOC, SECTIONS("/dept/description") "marketing") = 1')//employee
  return $i/name
```

在此示例中，使用全文本搜索的 SQL 查询的结果返回至 XQuery FLWOR 表达式的 for 子句。然后，for 子句返回所有 <employee> 元素，并且 return 子句返回检索到的 <employee> 元素中的 <name> 元素。

有关 DB2 Net Search Extender 的更多信息，请参阅 DB2 Net Search Extender 文档或 www.ibm.com/software/data/db2/extenders/netsearch。

将 XML 列中的数据检索至较早版本的 DB2 客户机

如果将数据从 XML 列检索到 DB2 版本 9.1 之前的发行版的客户机，那么数据库客户机不能处理 XML 数据。在 DRDA® 处理期间，当数据库服务器认识到客户机不能支持 XML 数据时，缺省情况下，DB2 数据库服务器将 XML 数据值描述为 BLOB 值并将数据作为 BLOB 数据发送至客户机。BLOB 数据是 XML 数据的已序列化字符串表示，并具有完整的 XML 声明。

如果想要将数据作为除 BLOB 数据类型之外的数据类型接收，那么使用下列方法之一:

- 要将数据作为 CLOB 数据检索，请数据库服务器的管理员使用 db2set 命令将服务器上的 DB2_MAP_XML_AS_CLOB_FOR_DLC 注册表变量设置为 YES。

要点：在数据库服务器上将 DB2_MAP_XML_AS_CLOB_FOR_DLC 注册表变量设置为 YES 时，**所有**处于较早发行版级别且连接至实例内的任何数据库的 DB2 客户机都接收 XML 数据作为 CLOB 数据。

要点：在数据库服务器上将 DB2_MAP_XML_AS_CLOB_FOR_DLC 注册表变量设置为 YES 时，客户机接收作为 XML 数据的已序列化字符串表示的 CLOB 数据，**但不具有** XML 声明。

- 要将数据作为 CLOB、CHAR 或 VARCHAR 数据检索，在 DB2 数据库服务器将数据发送至客户机之前，对列数据调用 XMLSERIALIZE 函数，以指示 DB2 数据库服务器将数据转换为指定的数据类型。

如果未调用 XMLSERIALIZE 来将数据从数据库服务器检索至较早发行版级别的客户机，那么从其检索数据的列的行为与检索 BLOB 或 CLOB 列的行为不完全相同。例如，虽然您可以对 BLOB 列使用 LIKE 谓词，但不能对返回 BLOB 或 CLOB 数据的 XML 列使用 LIKE 谓词。

用于构造 XML 值的 SQL/XML 发布函数

可通过组合与想要包括在生成的 XML 值中的组件对应的函数来构造 XML 值（该值不必是格式良好的 XML 文档）。必须按想要结果出现的顺序指定函数。

将使用 SQL/XML 发布函数构造的值返回为 XML。根据想要对 XML 值执行的操作，您可能需要显式序列化该值以将它转换为另一种 SQL 数据类型。有关详细信息，请参阅有关 XML 序列化的文档。

可使用下列 SQL/XML 发布函数来构造 XML 值。有关每个函数的语法描述，请参阅第 399 页的附录 B，『SQL/XML 发布函数』：

XMLAGG 聚集函数
: 返回一个 XML 序列，对于 XML 值集合中的每个非空值，该序列都包含一项。

XMLATTRIBUTES 标量函数
: 通过自变量构造 XML 属性。此函数只能用作 XMLELEMENT 函数的自变量。

XMLCOMMENT 标量函数
: 返回具有单个 XQuery 注释节点的 XML 值，该注释节点将输入自变量作为内容。

XMLCONCAT 标量函数
: 返回一个序列，该序列包含数目不定的 XML 输入自变量的并置。

XMLDOCUMENT 标量函数
: 返回具有单个 XQuery 文档节点的 XML 值，该文档节点有一个或多个子节点。此函数创建一个文档节点，根据定义，每个 XML 文档都必须有一个文档节点。文档节点在序列化的 XML 表示中不可视，但是，要存储在 DB2 表中的每个文档必须包含文档节点。

XMLELEMENT 标量函数
: 返回作为 XML 元素节点的 XML 值。请注意，XMLELEMENT 函数不创建文档节点，只创建元素节点。在构造要插入的 XML 文档时，仅创建元素节点是不够的。文档必须包含使用 XMLDOCUMENT 函数创建的文档节点。

XMLFOREST 标量函数
: 返回作为 XML 元素节点的序列的 XML 值。

XMLGROUP 聚集函数
: 返回单个顶级元素以表示一个表或查询结果。缺省情况下，结果集中的每行映射至行子元素，而每个输入表达式映射至行子元素的子元素。（可选）结果中的每行可映射至行子元素，每个输入表达式可映射至行子元素的属性。

XMLNAMESPACES 声明
: 通过自变量构造名称空间声明。此声明只能用作 XMLELEMENT、XMLFOREST 和 XMLTABLE 函数的自变量。

XMLPI 标量函数
: 返回具有单个 XQuery 处理指令节点的 XML 值。

XMLROW 标量函数
: 返回行元素序列以表示一个表或查询结果。缺省情况下，每个输入表达式变换为行元素的子元素。（可选）每个输入表达式可变换为行元素的属性。

XMLTEXT 标量函数
: 返回具有单个 XQuery 文本节点的 XML 值，该文本节点将输入自变量作为内容。

XSLTRANSFORM 标量函数
: 将 XML 数据转换为其他格式，包括其他 XML 模式。

空元素值

使用 XMLELEMENT 或 XMLFOREST 构造 XML 值时，在确定元素的内容时可能遇到空值。XMLELEMENT 和 XMLFOREST 的 EMPTY ON NULL 和 NULL ON NULL 选项允许您指定在元素的内容为空时是生成空元素还是不生成元素。XMLEXISTS 的缺省空值处理方式是 EMPTY ON NULL。XMLFOREST 的缺省空值处理方式是 NULL ON NULL。

发布 XML 值的示例

以下示例显示如何使用 SQL/XML 发布函数和 XQuery 表达式来构造 XML 值。

示例：使用常量值构造 XML 文档

此简单示例显示如何使用 SQL/XML 发布函数构造适合发布的常量 XML 值。

作为一个简单示例，请考虑以下 XML 元素：

```
<elem1 xmlns="http://posample.org" id="111">
  <!-- example document -->
    <child1>abc</child1>
    <child2>def</child2>
</elem1>
```

文档包括：

- 三个元素节点（elem1、child1 和 child2）
- 名称空间声明
- <elem1> 的“id”属性
- 注释节点

要构造此文档，执行下列步骤：

1. 使用 XMLELEMENT 创建名为“elem1”的元素节点。

2. 使用 XMLNAMESPACES 将缺省名称空间声明添加至 <elem1> 的 XMLELEMENT 函数调用。
3. 使用 XMLATTRIBUTES 创建名为“id”的属性，并将该属性放置在 XMLNAMESPACES 声明后面。
4. 使用 XMLCOMMENT 在 <elem1> 的 XMLELEMENT 函数调用中创建注释节点。
5. 使用 XMLFOREST 函数在 <elem1> 的 XMLELEMENT 函数调用内创建名为“child1”和“child2”的元素森林。

这些步骤组合成以下查询:

```
VALUES XMLELEMENT (NAME "elem1",
                   XMLNAMESPACES (DEFAULT 'http://posample.org'),
                   XMLATTRIBUTES ('111' AS "id"),
                   XMLCOMMENT ('example document'),
                   XMLFOREST('abc' as "child1",
                             'def' as "child2"))
```

示例: 使用单个表中的值构造 XML 文档

此示例显示如何使用 SQL/XML 发布函数通过单个表构造适合发布的 XML 值。

此示例说明如何通过存储在单个表中的值构造 XML 文档。在以下查询中，通过使用 XMLELEMENT 函数用 PRODUCT 表的 name 列中的值构造每个 <item> 元素。然后，使用 XMLAGG 在构造的 <allProducts> 元素内聚集所有 <item> 元素。

```
SELECT XMLELEMENT (NAME "allProducts",
                   XMLAGG(XMLELEMENT (NAME "item", p.name)))
FROM Product p
```

```
<allProducts>
  <item>Snow Shovel, Basic 22 inch</item>
  <item>Snow Shovel, Deluxe 24 inch</item>
  <item>Snow Shovel, Super Deluxe 26 inch</item>
  <item>Ice Scraper, Windshield 4 inch</item>
</allProducts>
```

通过使用 XMLROW 函数（而不是使用 XMLAGG 聚集元素），可构造包含行元素序列的类似 XML 文档。

```
SELECT XMLELEMENT (NAME "products",
                   XMLROW(NAME as "po:item"))
FROM Product
```

生成的输出如下所示:

```
<products>
   <row>
      <po:item>Snow Shovel, Basic 22 inch</po:item>
   </row>
</products>
<products>
   <row>
      <po:item>Snow Shovel, Deluxe 24 inch</po:item>
   </row>
</products>
<products>
      <row><po:item>Snow Shovel, Super Deluxe 26 inch</po:item>
   </row>
</products>
<products>
      <row><po:item>Ice Scraper, Windshield 4 inch</po:item>
```

```
  </row>
</products>

  4 record(s) selected.
```

示例：使用多个表中的值构造 XML 文档

此示例显示如何使用 SQL/XML 发布函数通过多个表构造适合发布的 XML 值。

此示例显示如果通过存储在多个表中的值构造 XML 文档。在以下查询中，使用 XMLFOREST 函数从名为 name 和 numInStock 的一组元素构造 <prod> 元素。此森林是使用 PRODUCT 和 INVENTORY 表中的值构造的。然后，在已构造的 <saleProducts> 元素内聚集所有 <prod> 元素。

```
SELECT XMLELEMENT (NAME "saleProducts",
                   XMLAGG (XMLELEMENT (NAME "prod",
                           XMLATTRIBUTES (p.Pid AS "id"),
                           XMLFOREST (p.name as "name",
                                      i.quantity as "numInStock"))))
FROM PRODUCT p, INVENTORY i
WHERE p.Pid = i.Pid
```

上一个查询产生以下 XML 文档：

```
<saleProducts>
  <prod id="100-100-01">
    <name>Snow Shovel, Basic 22 inch</name>
    <numInStock>5</numInStock>
  </prod>
  <prod id="100-101-01">
    <name>Snow Shovel, Deluxe 24 inch</name>
    <numInStock>25</numInStock>
  </prod>
  <prod id="100-103-01">
    <name>Snow Shovel, Super Deluxe 26 inch</name>
    <numInStock>55</numInStock>
  </prod>
  <prod id="100-201-01">
    <name>Ice Scraper, Windshield 4 inch</name>
    <numInStock>99</numInStock>
  </prod>
</saleProducts>
```

示例：使用包含空元素的表行中的值构造 XML 文档

此示例显示如何使用 SQL/XML 发布函数通过包含空元素的表行构造适合发布的 XML 值。

此示例假定 INVENTORY 表的 LOCATION 列在一行中包含空值。因此，以下查询不返回 <loc> 元素，因为缺省情况下 XMLFOREST 将空视为空：

```
SELECT XMLELEMENT (NAME "newElem",
                   XMLATTRIBUTES (PID AS "prodID"),
                   XMLFOREST (QUANTITY as "quantity",
                              LOCATION as "loc"))
FROM INVENTORY
```

```
<newElem prodID="100-100-01"><quantity>5</quantity></newElem>
```

指定了 EMPTY ON NULL 选项的相同查询返回一个空 <loc> 元素：

```
SELECT XMLELEMENT (NAME "newElem",
                   XMLATTRIBUTES (PID AS "prodID"),
                   XMLFOREST (QUANTITY as "quantity",
                              LOCATION as "loc" OPTION EMPTY ON NULL))
FROM INVENTORY
```

```
<newElem prodID="100-100-01"><quantity>5</quantity><loc /></newElem>
```

示例: 使用 XQuery 发布数据

此示例显示如何使用 SQL/XML 发布函数和 XQuery 表达式来构造适合发布的 XML 值。

以下 XQuery 表达式使用删除更新表达式来创建简单的客户列表。该表达式会从客户信息中除去客户标识、地址、辅助信息和非工作电话号码，并将 `address` 节点元素中的 `country` 属性移至 `customerinfo` 节点元素。

```
  xquery
<phonelist>
  {for $d in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
  return
    transform
    copy $mycust := $d
    modify (
      do delete ( $mycust/@Cid ,
        $mycust/addr ,
        $mycust/assistant ,
        $mycust/phone[@type!="work"] ),
      do insert attribute country { $mycust/addr/@country } into $mycust )
    return $mycust }
</phonelist>
```

注意，尽管删除了 `address` 元素，但可在 **modify** 子句中访问地址信息，并且插入表达式会使用 `address` 元素中的 `country` 属性。

查询将返回以下结果:

```
<phonelist>
 <customerinfo country="Canada">
<name>Kathy Smith</name>
<phone type="work">416-555-1358</phone>
 </customerinfo>
 <customerinfo country="Canada">
<name>Kathy Smith</name>
<phone type="work">905-555-7258</phone>
 </customerinfo>
 <customerinfo country="Canada">
<name>Jim Noodle</name>
<phone type="work">905-555-7258</phone>
 </customerinfo>
 <customerinfo country="Canada">
  <name>Robert Shoemaker</name>
<phone type="work">905-555-7258</phone>
 </customerinfo>
 <customerinfo country="Canada">
<name>Matt Foreman</name>
<phone type="work">905-555-4789</phone>
 </customerinfo>
 <customerinfo country="Canada">
<name>Larry Menard</name>
  <phone type="work">905-555-9146</phone>
 </customerinfo>
</phonelist>
```

使用 XSLT 样式表进行变换

将 XML 数据变换为其他格式的标准方法是通过可扩展样式表语言变换（XSLT）进行。可使用内置 XSLTRANSFORM 函数将 XML 文档转换为 HTML、纯文本或不同 XML 模式。

XSLT 使用样式表将 XML 转换为其他数据格式。可转换 XML 文档的部分或全部，并使用 XPath 查询语言和 XSLT 的内置函数选择或重新排列数据。XSLT 通常用于将 XML 转换为 HTML，但还可用于将符合一个 XML 模式的 XML 文档变换为符合另一个模式的文档。 XSLT 还可用于将 XML 数据转换为无关格式，如用逗号定界的文本或 troff 之类的格式化语言。XSLT 主要有两个用途:

- 格式化（将 XML 转换为 HTML 或 FOP 之类的格式化语言）；
- 数据交换（查询、重组以及将数据从一个 XML 模式转换为另一个 XML 模式，或者转换为 SOAP 之类的数据交换格式）。

两种情况都可能要求变换整个 XML 文档或仅变换所选部分。XSLT 包含 XPath 规范，允许查询并检索源 XML 文档中的任意数据。XSLT 模板还可能包含其他信息，如添加至输出文件的文件头和指令块。

XSLT 的工作方式

XSLT 样式表是用可扩展样式表语言（XSL，一种 XML 模式）编写的。XSL 是不同于 C 或 Perl 之类的算法语言的模板语言，此功能限制 XSL 的能力但使其非常适合其用途。XSL 样式表包含一个或多个 `template` 元素，它们描述在目标文件中遇到给定 XML 元素或查询时要采取的操作。典型 XSLT 模板元素将通过指定应用元素来启动。例如，

```
<xsl:template match="product">
```

声明此模板的内容将用于替换目标 XML 文件中遇到的任何 `<product>` 标记的内容。XSLT 文件由以非必要顺序列示的此类模板的列表组成。

以下示例显示 XSLT 模板的典型元素。在此情况下，目标为包含库存信息的 XML 文档，如描述雪刮的以下记录:

```
<?xml version="1.0"?>
<product pid="100-201-01">
   <description>
      <name>Ice Scraper, Windshield 4 inch</name>
      <details>Basic Ice Scraper 4 inches wide, foam handle</details>
      <price>3.99</price>
   </description>
</product>
```

此记录包括防风雪刮的部件号、描述和价格之类的信息。其中某些信息包含在 `<name>` 之类的元素中。部件号之类的信息包含在属性（在此情况下为 `<product>` 元素的 `pid` 属性）中。要将此信息显示为 Web 页面，可应用以下 XSLT 模板:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
   <xsl:template match="/">
       <html>
            <body>
                <h1><xsl:value-of select="/product/description/name"/></h1>
                <table border="1">
                        <th>
       <xsl:apply-templates select="product"/>
```

```
                    </th>
                 </table>
            </body>
        </html>
   </xsl:template>
     <xsl:template match="product">
        <tr>
                   <td width="80">product ID</td>
                   <td><xsl:value-of select="@pid"/></td>
              </tr>
              <tr>
                   <td width="200">product name</td>
                   <td><xsl:value-of select="/product/description/name"/></td>
              </tr>
              <tr>
                   <td width="200">price</td>
                   <td>$<xsl:value-of select="/product/description/price"/></td>
              </tr>
              <tr>
                   <td width="50">details</td>
                   <td><xsl:value-of select="/product/description/details"/></td>
              </tr>
  </xsl:template>
</xsl:stylesheet>
```

XSLT 处理器将以上模板和目标文档作为输入接收，它会输出以下 HTML 文档:

```
<html>
<body>
<h1>Ice Scraper, Windshield 4 inch</h1>
<table border="1">
<th>
<tr>
<td width="80">product ID</td><td>100-201-01</td>
</tr>
<tr>
<td width="200">product name</td><td>Ice Scraper, Windshield 4 inch</td>
</tr>
<tr>
<td width="200">price</td><td>$3.99</td>
</tr>
<tr>
<td width="50">details</td><td>Basic Ice Scraper 4 inches wide, foam handle</td>
</tr>
</th>
</table>
</body>
</html>
```

XSLT 处理器针对给定条件（通常每个模板一个条件）测试入局 XML 文档。如果满足条件，那么模板内容将插入到输出中；如果不满足条件，那么模板将通过处理器传递。样式表可将它自己的数据添加至输出，例如，在 HTML 表标记和“product ID”之类的字符串中。

XPath 可用于定义模板条件（例如，在 <xsl:template match="product"> 中）以及在 XML 流中（例如，在 <h1><xsl:value-of select="/product/description/name"/></h1> 中）选择并插入数据。

使用 XSLTRANSFORM

可使用 XSLTRANSFORM 函数来对 XML 数据应用 XSLT 样式表。如果提供函数、XML 文档的名称和 XSLT 样式表的名称， 那么该函数将对文档应用样式表并返回结果。

在运行时将参数传递至 XSLT 样式表

可在运行时使用内置 XSLTRANSFORM 函数传递参数来转换 XML 文档。

XSLTRANSFORM 函数的一个重要功能就是能够在运行时接受 XSLT 参数。如果没有此功能，那么需要维护大型 XSLT 样式表库（针对 XML 数据的查询的每个变体一个）；或者需要为每种新查询编辑样式表。参数传递允许您设计可单独保留的一般样式表，从而累积参数文件库或在使用时潜在构建。

XSLT 参数包含在单独的 XML 文档中，例如:

```
<?xml version="1.0"?>
<params xmlns="http://www.ibm.com/XSLTransformParameters">
        <param name="headline">BIG BAZAAR super market</param>
        <param name="supermarketname" value="true"/>
</params>
```

每个 `<param>` 元素命名一个参数，并在 `value` 属性中包含其值（对于较长的值，则包含在元素本身中）。上述示例显示两个变体。

XSLT 模板文件允许的参数是使用 `<xsl:param>` 元素作为变量定义的，如下所示:

```
<xsl:param name="headline"/>
<xsl:param name="supermarketname"/>
```

在此示例中，可在样式表内的任何位置调用 `$headline` 或 `$supermarketname` 变量，并且它们将包含参数文件中定义的文件（在此情况下，分别为字符串“BIG BAZAAR super market”及值“true”）。

示例: 将 XSLT 用作格式化引擎

一个示例，演示如何将内置 XSLTRANSFORM 函数用作格式化引擎。

此示例演示如何将 XSLT 用作格式化引擎。要进行设置，先将下面两个示例文档插入到数据库中。

```
INSERT INTO XML_TAB VALUES
(1,
    '<?xml version="1.0"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation = "/home/steffen/xsd/xslt.xsd">
<student studentID="1" firstName="Steffen" lastName="Siegmund"
    age="23" university="Rostock"/>
</students>',
    '<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="showUniversity"/>
<xsl:template match="students">
     <html>
              <head/>
                 <body>
                  <h1><xsl:value-of select="$headline"/></h1>
                  <table border="1">
                    <th>
                        <tr>
                                <td width="80">StudentID</td>
                                <td width="200">First Name</td>
                                <td width="200">Last Name</td>
                                <td width="50">Age</td>
                                <xsl:choose>
       <xsl:when test="$showUniversity ='true'">
```

```
                                   <td width="200">University</td>
                                    </xsl:when>
       </xsl:choose>
                             </tr>
                          </th>
                          <xsl:apply-templates/>
                  </table>
                 </body>
                </html>
         </xsl:template>
        <xsl:template match="student">
              <tr>
                  <td><xsl:value-of select="@studentID"/></td>
                  <td><xsl:value-of select="@firstName"/></td>
                  <td><xsl:value-of select="@lastName"/></td>
                  <td><xsl:value-of select="@age"/></td>
                  <xsl:choose>
                      <xsl:when test="$showUniversity = 'true' ">
                        <td><xsl:value-of select="@university"/></td>
                      </xsl:when>
                  </xsl:choose>
       </tr>
        </xsl:template>
</xsl:stylesheet>'
);
```

下面调用 XSLTRANSFORM 函数以将 XML 数据转换为 HTML 并显示出来。

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC AS CLOB(1M)) FROM XML_TAB;
```

结果为以下文档:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">First Name</td>
<td width="200">Last Name</td>
<td width="50">Age</td>
</tr>
</th>
 <tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>23</td>
</tr>
  </table>
</body>
</html>
```

在此示例中，输出为 HTML 并且这些参数仅影响产生的 HTML 内容及提供的数据。这样它会演示如何将 XSLT 用作最终用户输出的格式化引擎。

示例: 使用 XSLT 来进行数据交换

一个示例，演示如何使用内置 XSLTRANSFORM 函数转换 XML 文档来进行数据交换。

此示例演示如何通过使用参数和样式表在运行时产生不同数据交换格式。

使用包含 xsl:param 元素的样式表捕获参数文件中的数据。

```
INSERT INTO Display_productdetails values(1, '<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="supermarketname"/>
<xsl:template match="product">
   <html>
             <head/>
                <body>
                <h1><xsl:value-of select="$headline"/></h1>
                <table border="1">
                        <th>
                        <tr>
                         <td width="80">product ID</td>
                         <td width="200">product name</td>
                         <td width="200">price</td>
                         <td width="50">details</td>
                         <xsl:choose>
   <xsl:when test="$supermarket =''true'' ">
                                 <td width="200">BIG BAZAAR super market</td>
                                </xsl:when>
    </xsl:choose>
                         </tr>
                        </th>
                        <xsl:apply-templates/>
                </table>
                </body>
                </html>
        </xsl:template>
      <xsl:template match="product">
             <tr>
                   <td><xsl:value-of select="@pid"/></td>
                   <td><xsl:value-of select="/product/description/name"/></td>
                   <td><xsl:value-of select="/product/description/price"/></td>
                   <td><xsl:value-of select="/product/description/details"/></td>
                  </tr>
     </xsl:template>
  </xsl:stylesheet>'
);
```

该参数文件包含对应于 XSLT 模板中的参数的参数及内容:

```
CREATE TABLE PARAM_TAB (DOCID INTEGER, PARAM VARCHAR (10K));

INSERT INTO PARAM_TAB VALUES
(1,
'<?xml version="1.0"?>
<params xmlns="http://www.ibm.com/XSLTransformParameters">
        <param name="supermarketname" value="true"/>
        <param name="headline">BIG BAZAAR super market</param>
</params>'
);
```

然后，可使用以下命令在运行时应用参数文件:

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC WITH PARAM AS CLOB (1M))
   FROM product_details X, PARM_TAB P WHERE X.DOCID=P.DOCID;
```

结果为 HTML，但包含由参数文件确定的内容及对 XML 文档内容执行的测试:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
```

```
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">product ID</td>
<td width="200">product Name</td>
<td width="200">price</td>
<td width="50">Details</td>
</tr>
</th>
</table>
</body>
</html>
```

在其他应用程序中，XSLTRANSFORM 的输出不能为 HTML，但可以是另一 XML 文档或使用另一数据格式的文件，如 EDI 文件。

对于数据交换应用程序，参数文件可包含 EDI 或 SOAP 文件头信息（如电子邮件或端口地址），或对特定事务唯一的其他关键数据。因为上述示例中使用的 XML 是库存记录，所以可以轻松想像如何使用 XSLT 重新打包此记录以便与客户端的订购系统交换数据。

示例：使用 XSLT 来除去名称空间

您接收到的 XML 文档可能包含不需要或者不正确的名称空间信息。可以使用 XSLT 样式表来除去或处理文档中的名称空间信息。

下列示例说明如何使用 XSLT 来除去 XML 文档中的名称空间信息。这些示例将 XML 文档和 XSLT 样式表存储在 XML 列中，并使用 XSLTRANSFORM 函数且使用其中一种 XSLT 样式表来转换 XML 文档。

下列 CREATE 语句将创建 XMLDATA 和 XMLTRANS 这两个表。XMLDATA 表中包含一个样本 XML 文档，XMLTRANS 表中包含 XSLT 样式表。

```
CREATE TABLE XMLDATA (ID BIGINT NOT NULL PRIMARY KEY, XMLDOC XML );
CREATE TABLE XMLTRANS (XSLID BIGINT NOT NULL PRIMARY KEY, XSLT XML );
```

使用以下 INSERT 语句将样本 XML 文档添加至 XMLDATA 表。

```
insert into XMLDATA (ID, XMLDOC) values ( 1, '
<newinfo xmlns="http://mycompany.com">
<!-- merged customer information -->
  <customerinfo xmlns="http://oldcompany.com" xmlns:d="http://test" Cid="1004">
     <name>Matt Foreman</name>
  <addr country="Canada">
     <street>1596 Baseline</street>
     <city>Toronto</city>
     <prov-state>Ontario</prov-state>
     <pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
  <phone type="work">905-555-4789</phone>
  <h:phone xmlns:h="http://test1" type="home">416-555-3376</h:phone>
  <d:assistant>
      <name>Gopher Runner</name>
      <h:phone xmlns:h="http://test1" type="home">416-555-3426</h:phone>
  </d:assistant>
  </customerinfo>
</newinfo>
');
```

用于除去所有名称空间的示例 XSLT 样式表

以下示例使用 XSLT 样式表从存储在 XMLDATA 表内的 XML 文档中除去所有名称空间信息。这些示例将样式表存储在 XMLTRANS 表中，并使用 SELECT 语句将此样式表应用于 XML 文档。

使用 INSERT 语句将此样式表添加至 XMLTRANS 表。

```
insert into XMLTRANS (XSLID, XSLT) values ( 1, '
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

 <!-- keep comments -->
 <xsl:template match="comment()">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="*">
    <!-- remove element prefix -->
    <xsl:element name="{local-name()}">
      <!-- process attributes -->
      <xsl:for-each select="@*">
        <!-- remove attribute prefix -->
        <xsl:attribute name="{local-name()}">
          <xsl:value-of select="."/>
        </xsl:attribute>
      </xsl:for-each>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
') ;
```

以下 SELECT 语句使用 XSLT 样式表来转换样本 XML 文档。

```
SELECT XSLTRANSFORM (XMLDOC USING XSLT ) FROM XMLDATA, XMLTRANS
   WHERE ID = 1 and XSLID = 1
```

XSLTRANSFORM 命令使用第一个 XSLT 样式表来转换 XML 文档，将返回以下 XML 并除去所有名称空间信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<newinfo>
 <!-- merged customer information -->
<customerinfo Cid="1004">
<name>Matt Foreman</name>
<addr country="Canada">
<street>1596 Baseline</street>
<city>Toronto</city>
<prov-state>Ontario</prov-state>
<pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
<phone type="work">905-555-4789</phone>
<phone type="home">416-555-3376</phone>
  <assistant>
<name>Gopher Runner</name>
<phone type="home">416-555-3426</phone>
  </assistant>
 </customerinfo>
</newinfo>
```

用于保留元素的名称空间绑定的示例 XSLT 样式表

以下示例使用 XSLT 样式表来仅保留 phone 元素节点的名称空间绑定。此节点的名称在 XSLT 变量 mynode 中指定。这些示例将样式表存储在 XMLTRANS 表中，并使用 SELECT 语句将此样式表应用于 XML 文档。

使用以下 INSERT 语句将此样式表添加至 XMLTRANS 表。

```
insert into XMLTRANS (XSLID, XSLT) values ( 2, '
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:variable name ="mynode">phone</xsl:variable>

  <!-- keep comments -->
  <xsl:template match="comment()">
      <xsl:copy>
        <xsl:apply-templates/>
      </xsl:copy>
   </xsl:template>

  <xsl:template xmlns:d="http://test" xmlns:h="http://test1" match="*">
  <xsl:choose>

  <!-- keep namespace prefix for node names $mynode -->
    <xsl:when test="local-name() = $mynode " >
     <xsl:element name="{name()}">
       <!-- process node attributes -->
       <xsl:for-each select="@*">
       <!-- remove attribute prefix  -->
         <xsl:attribute name="{local-name()}">
           <xsl:value-of select="."/>
         </xsl:attribute>
       </xsl:for-each>
      <xsl:apply-templates/>
     </xsl:element>
    </xsl:when>

  <!-- remove namespace prefix from node -->
    <xsl:otherwise>
     <xsl:element name="{local-name()}">
       <!-- process node attributes -->
       <xsl:for-each select="@*">
       <!-- remove attribute prefix  -->
         <xsl:attribute name="{local-name()}">
           <xsl:value-of select="."/>
         </xsl:attribute>
       </xsl:for-each>
      <xsl:apply-templates/>
     </xsl:element>
     </xsl:otherwise>
  </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
');
```

以下 SELECT 语句使用第二个 XSLT 样式表来转换样本 XML 文档。

```
SELECT XSLTRANSFORM (XMLDOC USING XSLT) FROM XMLDATA, XMLTRANS
   WHERE ID = 1 and XSLID = 2 ;
```

XSLTRANSFORM 命令使用第二个 XSLT 样式表来转换 XML 文档，并返回以下 XML 以及仅 phone 元素的名称空间。

```
<?xml version="1.0" encoding="UTF-8"?>
<newinfo>
 <!-- merged customer information -->
<customerinfo Cid="1004">
<name>Matt Foreman</name>
<addr country="Canada">
<street>1596 Baseline</street>
<city>Toronto</city>
<prov-state>Ontario</prov-state>
<pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
<phone type="work">905-555-4789</phone>
  <h:phone xmlns:h="http://test1" type="home">
   416-555-3376
  </h:phone>
  <assistant>
<name>Gopher Runner</name>
   <h:phone xmlns:h="http://test1" type="home">
    416-555-3426
   </h:phone>
  </assistant>
 </customerinfo>
</newinfo>
```

变换 XML 文档的重要注意事项

使用内置 XSLTRANSFORM 函数转换 XML 文档时，有一些重要注意事项和限制。

变换 XML 文档时注意下列事项:

- 源 XML 文档必须是单一根并且格式良好。
- 因为 XSLT 变换在缺省情况下会产生 UTF-8 字符，所以插入到使用字符数据类型定义的列中时，输出流会丢失字符。

限制

- 仅支持 W3C XSLT V1.0 建议。
- 所有参数和结果类型必须为 SQL 类型，它们不能是文件名。
- 不支持包含多个样式表文档的变换（使用 xsl:include 声明）。

使用说明

XSLTRANSFORM 函数未打算用于高性能应用程序，并且无法替换应用程序服务器中的类似功能。

SQL/XML 发布函数中的特殊字符处理

SQL/XML 发布函数在处理特殊字符时具有缺省行为。

SQL 值至 XML 值

某些字符在 XML 文档内被视为特殊字符，必须使用这些字符的实体表示用已转义格式显示。这些特殊字符如下所示:

表 22. 特殊字符及其实体表示

特殊字符	实体表示
<	<
>	>

表 22. 特殊字符及其实体表示 (续)

特殊字符	实体表示
&	&
"	"

使用 SQL/XML 发布函数将 SQL 值作为 XML 值发布时，将对这些特殊字符进行转义并将它们替换为其预定义的实体。

SQL 标识和 QName

通过 SQL 值发布或构造 XML 值时，可能必须将 SQL 标识映射至 XML 限定名或 QName。但是，定界 SQL 标识中允许的字符集与 QName 中允许的字符集不同。这种差别表示在 SQL 标识中使用的一些字符在 QName 中无效。因此，这些字符将替换为 QName 中它们的实体表示。

例如，考虑定界 SQL 标识“phone@work”。因为 @ 字符在 QName 中是无效字符，所以将对该字符进行转义，QName 将成为：phone@work。

请注意，此缺省转义行为仅适用于列名。对于作为 XMLELEMENT 中的元素名称提供的 SQL 标识，或作为 XMLFOREST 和 XMLATTRIBUTES 的 AS 子句中的别名提供的 SQL 标识，没有转义缺省值。在这些情况下，必须提供有效的 QName。有关有效名称的更多详细信息，请参阅 W3C XML 名称空间规范。

XML 序列化

XML 序列化是将 XML 数据从它在 XQuery 和 XPath 数据模型中的表示（它在 DB2 数据库中的分层格式）转换为它在应用程序中的序列化字符串格式的过程。

可以让 DB2 数据库管理器隐式执行序列化，也可以调用 XMLSERIALIZE 函数来显式请求 XML 序列化。将 XML 数据从数据库服务器发送至客户机时，通常要使用 XML 序列化。

在大多数情况下，隐式序列化是首选方法，因为它编码较简单，并且将 XML 数据发送至客户机时允许 DB2 客户机正确处理 XML 数据。显式序列化需要如下所述的其他处理，这些处理在隐式序列化期间由客户机自动处理。

通常，首选隐式序列化，因为在将数据作为 XML 数据发送至客户机时它更有效。但是，在下列情况下，最好进行显式 XMLSERIALIZE:

最好将 XML 数据转换为 BLOB 数据类型，因为检索二进制数据会导致较少的编码问题。

隐式 XML 序列化

借助隐式序列化，在客户机支持 XML 数据类型的情况下，将数据发送至客户机时数据具有 XML 类型。对于DB2 CLI和嵌入式 SQL 应用程序，DB2 数据库服务器将使用适当编码规范的 XML 声明添加至数据。对于 Java 和 .NET 应用程序，DB2 数据库服务器不添加 XML 声明，但如果将数据检索到 `DB2Xml` 对象中并使用某些方法来检索该对象中的数据，那么IBM 数据服务器 JDBC 和 SQLJ 驱动程序会添加 XML 声明。

示例: 在 C 程序中，隐式序列化客户标识为 '1000' 的 customerinfo 文档并将序列化的文档检索到二进制 XML 主变量中。检索到的数据使用 UTF-8 编码方案，并且包含 XML 声明。

```
  EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS XML AS BLOB (1M) xmlCustInfo;
  EXEC SQL END DECLARE SECTION;
...
EXEC SQL SELECT INFO INTO :xmlCustInfo
    FROM Customer
    WHERE Cid=1000;
```

显式 XML 序列化

在显式调用 XMLSERIALIZE 之后，数据在数据库服务器中具有非 XML 数据类型，并且以该数据类型发送至客户机。

XMLSERIALIZE 允许您指定:

- 序列化时数据转换为的 SQL 数据类型

  该数据类型为字符或二进制数据类型。

- 输出数据是否应包括以下显式编码规范（EXCLUDING XMLDECLARATION 或 INCLUDING XMLDECLARATION）:

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  ```

来自 XMLSERIALIZE 的输出是 Unicode UTF-8 编码的数据。

如果将序列化的数据检索到非二进制数据类型中，那么该数据将转换为应用程序编码，但不修改编码规范。因此，数据的编码很可能不符合编码规范。这种情况会导致应用程序进程无法解析的 XML 数据依赖于编码名。

通常，首选隐式序列化，因为在将数据作为 XML 数据发送至客户机时它更有效。但是，在下列情况下，最好进行显式 XMLSERIALIZE:

- 当 XML 文档非常大时

  因为没有 XML 定位器，所以在 XML 文档非常大时，应使用 XMLSERIALIZE 将数据转换为 LOB 类型，以便可以使用 LOB 定位器。

- 当客户机不支持 XML 数据时

  如果客户机是不支持 XML 数据类型的较早版本，并且您使用隐式 XML 序列化，那么 DB2 数据库服务器在将数据发送至客户机之前将它转换为下列数据类型之一:

  – 缺省情况下，为 BLOB 数据类型

  – 如果在服务器上使用 db2set 命令将 DB2_MAP_XML_AS_CLOB_FOR_DLC 注册表变量设置为 YES，那么为 CLOB 数据类型

  如果想要检索到的数据是一些其他数据类型，那么可以执行 XMLSERIALIZE。

示例: 样本表 Customer 中的 XML 列 Info 包含一个文档，该文档包含等价于下列数据的分层:

```
<customerinfo Cid='1000'>
  <name>Kathy Smith</name>
  <addr country='Canada'>
  <street>5 Rosewood</street>
  <city>Toronto</city>
```

```
  <prov-state>Ontario</prov-state>
  <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-1358</phone>
</customerinfo>
```

调用 XMLSERIALIZE 以在将数据检索到主变量中之前序列化该数据并将它转换为 BLOB 类型。

```
SELECT XMLSERIALIZE(Info as BLOB(1M)) from Customer
  WHERE CID=1000
```

存储和检索后 XML 文档中的差别

将 XML 文档存储在 DB2 数据库中，然后从该数据库检索此副本后，检索到的文档可能与原始文档不完全相同。此行为由 XML 和 SQL/XML 标准定义，它与 Xerces 开放式源代码 XML 解析器的行为相同。

存储文档时会对该文档进行一些更改。这些更改包括:

- 如果执行 XMLVALIDATE，那么数据库服务器:
  - 将 XMLVALIDATE 调用中所指定的 XML 模式的缺省值添加到输入文档
  - 去掉输入文档中的可忽略空格
- 如果未请求 XML 验证，那么数据库服务器:
  - 去掉边界空格（如果未请求保留）
  - 按 XML 1.0 规范中所指定的执行行结尾规范化
  - 按 XML 1.0 规范中所指定的执行属性值规范化

    此过程导致属性中的换行符（U+000A）替换为空格字符（U+0020）。

从 XML 列中检索数据时会发生其他更改。这些更改包括:

- 如果在将数据发送至数据库服务器之前该数据具有 XML 声明，那么不保留 XML 声明。

  对于 DB2 CLI 和嵌入式 SQL 应用程序，通过隐式序列化，DB2 数据库服务器将使用适当编码规范的 XML 声明添加到数据。对于 Java 和 .NET 应用程序，DB2 数据库服务器不添加 XML 声明，但如果将数据检索到 DB2Xml 对象中并使用某些方法来检索该对象中的数据，那么IBM 数据服务器 JDBC 和 SQLJ 驱动程序会添加 XML 声明。

  如果执行 XMLSERIALIZE 函数，那么在指定 INCLUDING XMLDECLARATION 选项的情况下，DB2 数据库服务器将添加使用 UTF-8 编码的编码规范的 XML 声明。
- 在文档内容中或属性值中，某些字符将替换为它们的预定义 XML 实体。这些字符及其预定义实体为:

字符	Unicode 值	实体表示
AMPERSAND	U+0026	&
LESS-THAN SIGN	U+003C	<
GREATER-THAN SIGN	U+003E	>

- 在属性值或文本值内，某些字符将替换为它们的数字表示。这些字符及其数字表示为:

字符	Unicode 值	实体表示
CHARACTER TABULATION	U+0009	
LINE FEED	U+000A
CARRIAGE RETURN	U+000D
NEXT LINE	U+0085	…
LINE SEPARATOR	U+2028	

- 在属性值内，QUOTATION MARK (U+0022) 字符将替换为它的预定义 XML 实体 "。
- 如果输入文档具有 DTD 声明，那么不会保留该声明，并且不会生成基于 DTD 的标记。
- 如果输入文档包含 CDATA 部分，那么输出中不会保留这些部分。

归档 XML 文档的数据类型

虽然可以在任何二进制或字符类型的列中存储 XML 序列化字符串数据，但应该使用非 XML 列来归档 XML 数据。用于归档 XML 数据的最佳列数据类型是二进制数据类型，例如 BLOB。如果使用字符列来进行归档，就会进行代码页转换，这可能会使文档与其原始格式不一致。

第 6 章 为 XML 数据建立索引

XML 数据索引可用于提高查询存储在 XML 列中的 XML 文档的效率。

与索引键由指定的一个或多个表列组成的传统关系索引不同，XML 数据索引使用特定 XML 模式表达式对存储在单个列中的 XML 文档中的路径和值建立索引。该列的数据类型必须是 XML。

XML 数据索引根据 XML 模式表达式创建索引键，从而提供对文档内的节点的访问，而不是提供对文档开头的访问。因为可能 XML 文档中的多个部分都符合 XML 模式，所以可以将多个索引键插入一个文档的索引中。

可以使用 CREATE INDEX 语句创建XML 数据索引，使用 DROP INDEX 语句删除 XML 数据索引。随 CREATE INDEX 语句包括的 GENERATE KEY USING XMLPATTERN 子句指定想要建立索引的对象。

与 CREATE INDEX 语句一起用于非 XML 列的索引的一些关键字不适用于基于 XML 数据的索引。对于基于 XML 数据的索引，UNIQUE 关键字也有不同的含义。

示例: 创建XML 数据索引: 假定表 companyinfo 有一个称为 companydocs 的 XML 列，它包含诸如以下的 XML 文档片段:

Company1 的文档:

```
<company name="Company1">
  <emp id="31201" salary="60000" gender="Female">
    <name>
      <first>Laura</first>
      <last>Brown</last>
    </name>
    <dept id="M25">
      Finance
    </dept>
  </emp>
</company>
```

Company2 的文档:

```
<company name="Company2">
  <emp id="31664" salary="60000" gender="Male">
    <name>
      <first>Chris</first>
      <last>Murphy</last>
    </name>
    <dept id="M55">
      Marketing
    </dept>
  </emp>
  <emp id="42366" salary="50000" gender="Female">
    <name>
      <first>Nicole</first>
      <last>Murphy</last>
    </name>
    <dept id="K55">
```

```
      Sales
    </dept>
  </emp>
</company>
```

companyinfo 表的用户通常使用职员标识检索职员信息。可以使用诸如以下的索引使检索效率更高:

```
CREATE INDEX empindex on companyinfo(companydocs)
  GENERATE KEY USING XMLPATTERN '/company/emp/@id'
   AS SQL DOUBLE
```



图 5. XML 数据索引的示例

图 5 的注释:

1 XML 数据索引是对 companyinfo 表的 companydocs 列定义的。companydocs 必须是 XML 数据类型。

2 GENERATE KEY USING XMLPATTERN 子句提供关于想要建立索引的对象的信息。此子句称为 XML 索引规范。XML 索引规范包含 XML 模式子句。此示例中的 XML 模式子句指示想要对每个职员元素的标识属性值建立索引。

3 AS SQL DOUBLE 指示已建立索引的值存储为 DOUBLE 值。

索引 XML 模式表达式

只对存储在 XML 列中并满足 XML 模式表达式的那部分 XML 文档建立索引。要对 XML 模式建立索引，随 CREATE INDEX 语句一起提供索引规范子句。索引规范子句以 GENERATE KEY USING XMLPATTERN 开头，后跟 XML 模式和XML 数据索引的数据类型。或者，可以指定子句 GENERATE KEYS USING XMLPATTERN。

每个 CREATE INDEX 语句只允许有一个索引规范子句。可以对一个 XML 列创建多个 XML 索引。

XML 模式表达式

要标识将建立索引的那部分文档，可使用 XML 模式来指定 XML 文档内的节点集。此模式表达式与 XQuery 语言中定义的路径表达式类似，其差别在于前者仅支持一部分 XQuery 语言。

路径表达式步骤由正斜杠（/）分隔。还可以指定双正斜杠（//），它是 /descendant-or-self::node()/ 的缩写语法。在每个步骤中，选择正向轴（child::，@，attribute::，descendant::，self:: 和 descendant-or-self::），然后选择 XML 名称测试或 XML 种类测试。如果未指定正向轴，那么使用 child 轴作为缺省值。

如果使用 XML 名称测试，那么使用限定 XML 名称或通配符来指定与路径中的步骤匹配的节点名。XML 种类测试不匹配节点名，但它也可用来指定模式中要匹配的节点的种类：文本节点、注释节点、处理指令节点或任何其他类型的节点。

以下是一些不同模式表达式的示例。

```
1.  CREATE INDEX empindex on company(companydocs)
         GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

```
2.  CREATE INDEX empindex on company(companydocs)
         GENERATE KEY USING XMLPATTERN '/child::company/child::emp/attribute::id'
         AS SQL DOUBLE
```

```
3.  CREATE INDEX idindex on company(companydocs)
         GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
```

```
4.  CREATE INDEX idindex on company(companydocs)
         GENERATE KEY USING XMLPATTERN '/descendant-or-self::node()/attribute::id'
         AS SQL DOUBLE
```

```
5.  CREATE INDEX empindex on company(companydocs)
         GENERATE KEY USING XMLPATTERN '/company/emp/name/last/text()' AS SQL
         VARCHAR(25)
```

注: 语句 1 和语句 2 在逻辑上等效。语句 2 使用未缩写的语法。语句 3 和语句 4 在逻辑上等效。语句 4 使用未缩写的语法。

限定路径和节点

考虑一个名为“company”的表，其 XML 列（companydocs）中存储了 XML 文档。XML 文档具有两个路径的层次结构：“/company/emp/dept/@id”和“/company/emp/@id”。如果 XML 模式指定单个路径，那么该文档中的一组节点可能符合条件。

例如，如果用户希望在职员元素中搜索特定职员标识属性 (@id)，那么可以对 XML 模式“/company/emp/@id”创建索引。然后，使用格式为“/company/emp[@id=42366]”的谓词的查询可以利用 XML 列的索引。在此示例中，CREATE INDEX 语句中的 XMLPATTERN ’/company/emp/@id’ 指定引用文档中的许多不同节点的单个路径，因为文档中的每个职员元素都可能具有职员标识属性。

```
CREATE INDEX empindex on company(companydocs)
         GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

如果 XML 模式使用通配符表达式、descendant 轴或 descendant-or-self 轴，那么一组路径和节点可能符合条件。在以下示例中，指定了 descendant-or-self 轴，以便 XML 模式“//@id”引用部门标识属性和职员标识属性的路径，因为它们都包含 @id。

```
CREATE INDEX idindex on company(companydocs)
         GENERATE KEYS USING XMLPATTERN '//@id' AS SQL DOUBLE
```

XML 名称空间声明

限定 XML 名称（QName）用于定义 XML 模式表达式中的元素和属性标记。QName 的限定符是一个已经与名称空间 URI 相关联的名称空间前缀。

可以使用可选名称空间声明来指定 XML 模式，以将名称空间前缀映射至名称空间 URI 字符串文字，或者为 XML 模式定义缺省名称空间 URI。然后，将名称空间前缀用来限定 XML 模式中的元素和属性的名称，以使它们与文档中使用的名称空间相匹配。

在以下示例中，将缩写名称空间前缀 *m* 映射至 *http://www.mycompanyname.com/*

```
CREATE INDEX empindex on department(deptdocs)
          GENERATE KEYS USING XMLPATTERN
          'declare namespace m="http://www.mycompanyname.com/";
          /m:company/m:emp/m:name/m:last' AS SQL VARCHAR(30)
```

请注意，如果从命令行处理器（CLP）发出此 CREATE INDEX 语句，那么嵌入的分号将出现问题，因为分号是缺省语句终止符。为了避免此问题，使用下列变通方法之一：

- 确保分号不是一行中的最后一个非空格字符（例如，通过在分号后面添加空的 XQuery 注释）。
- 从命令行更改 CLP 中的缺省语句终止符。

还可以在同一个 XMLPATTERN 表达式中指定多个名称空间声明，但名称空间前缀在名称空间声明列表中必须唯一。此外，用户还可以选择为没有前缀的元素声明缺省名称空间。如果没有为元素显式指定名称空间或名称空间前缀，那么将使用缺省名称空间。缺省名称空间声明不适用于属性。如果用户未指定缺省名称空间，那么名称空间将为 *no namespace*。只能声明一个缺省名称空间。此名称空间声明行为遵从 XQuery 规则。

还可以使用缺省名称空间编写上一个示例:

```
CREATE INDEX empindex on department(deptdocs)
   GENERATE KEY USING XMLPATTERN
   'declare default element namespace "http://www.mycompany.com/";
    /company/emp/name/last') AS SQL VARCHAR(30)
```

在下一个示例中，*@id* 属性具有 *no namespace* 名称空间，这是因为缺省名称空间 *http://www.mycompany.com/* 仅适用于 *company* 和 *emp* 元素，但不适用于 *@id* 属性。这遵守基本 XQuery 规则，因为在 XML 文档中，缺省名称空间声明不适用于属性。

```
CREATE INDEX empindex on department(deptdocs)
   GENERATE KEY USING XMLPATTERN
   'declare default element namespace "http://www.mycompany.com/";
    /company/emp/@id' AS SQL VARCHAR(30)
```

由于 *@id* 属性应该具有与 *company* 和 *emp* 元素相同的名称空间，所以可以按如下所示编写语句:

```
CREATE INDEX empindex on department(deptdocs)
   GENERATE KEY USING XMLPATTERN
   'declare default element namespace "http://www.mycompany.com/";
    declare namespace m="http://www.mycompanyname.com/";
    /company/emp/@m:id' AS SQL VARCHAR(30)
```

用于创建索引的名称空间前缀与实例文档中使用的名称空间前缀不需要匹配即可建立索引，但完全扩展的 QName 需要匹配。前缀扩展为的名称空间的值（而不是前缀名本身）非常重要。例如，如果索引的名称空间前缀定义为 *m="http//www.mycompany.com/"*，并且实例文档中使用的名称空间前缀为 *c="http//www.mycompany.com/"*，那么将对实例文档中的 *c:company/c:emp/@id* 建立索引，因为缩写名称空间前缀 *m* 和 *c* 都扩展为同一名称空间。

与索引 XML 模式表达式关联的数据类型

在 CREATE INDEX 语句中指定的每个 XML 模式表达式必须与数据类型关联。支持以下四种 SQL 数据类型：VARCHAR、DATE、TIMESTAMP 和 DOUBLE。

可以选择将表达式的结果解释为多种数据类型。例如，值 *123* 具有字符表示，但也可以将它解释为数字 *123*。如果希望将路径 */company/emp/@id* 同时作为字符串和数字值来建立索引，那么必须创建两个索引，一个表示 VARCHAR 数据类型，一个表示 DOUBLE 数据类型。将文档中的值转换为对每个索引指定的数据类型。

以下示例说明如何对同一个 XML 列 *deptdocs* 创建两个具有不同数据类型的索引:

```
CREATE INDEX empindex1 on department(deptdocs)
         GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(10)

CREATE INDEX empindex2 on department(deptdocs)
       GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

描述受支持的 SQL 数据类型:

VARCHAR(*integer*)

采用 UTF-8 代码页将 VARCHAR 数据存储在 XML 列的索引中。如果将数据类型 VARCHAR 与指定的长度 *integer*（以字节为单位）配合使用，那么将指定的长度视为约束。如果在创建索引的同时将文档插入到表中或表中存在文档，那么在使用比指定长度要长的值对节点建立索引时，文档插入或索引创建将失败。如果插入或创建成功，那么可保证索引完整地存储所有字符串值，并且它同时支持范围扫描和相等查询。长度 *integer* 是范围在 1 到与页大小相关的最大值内的一个值。有关允许的最大长度的列表，请参阅 CREATE INDEX 语句。XQuery 语义用于字符串比较，其中的尾部空格非常重要。这与 SQL 语义不同，其中的尾部空格在比较期间不重要。

```
CREATE INDEX empindex1 on department(deptdocs)
     GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(50)
```

VARCHAR HASHED

可以指定 VARCHAR HASHED 来处理对任意长度的字符串建立索引。如果文档包含的要建立索引的字符串超出索引根据页大小相关的最大值所允许的最大长度 *integer*，那么可以取而指定 VARCHAR HASHED。在此示例中，系统对整个字符串生成 8 个字节的散列代码，并且对已建立索引的字符串的长度没有限制。如果指定 VARCHAR HASHED，那么无法执行范围扫描，因为索引包含散列码而不是实际的字符数据。只能将使用这些散列字符串的索引用于相等查询。XQuery 语义用于字符串相等比较，其中的尾部空格非常重要。这与 SQL 语义不同，其中的尾部空格在比较期间不重要。字符串散列保留 XQuery 相等语义，但不保留 SQL 相等语义。

```
CREATE INDEX empindex on company(companydocs)
    GENERATE KEY USING XMLPATTERN '/company/emp/name/last' AS SQL
    VARCHAR HASHED
```

DOUBLE

将转换所有数值并将它们作为 DOUBLE 数据类型存储在索引中。无限大的十进制类型和 64 位整数在作为 DOUBLE 存储时可能会丢失精度。索引 SQL 数据类型 DOUBLE 的值可能包括特殊数字值 *NaN*、*INF*、*-INF*、*+0* 和 *-0*，即使 SQL 数据类型 DOUBLE 本身不支持这些值。

DATE

在将 DATE 数据类型值存储在索引中之前，将它们规范化为 UTC（全球标准时间）或祖鲁时间。请注意，XML 模式数据类型的 DATE 允许的精度大于 SQL 数据类型。如果遇到超出范围的值，那么将返回错误。

TIMESTAMP

在将 TIMESTAMP 数据类型值存储在索引中之前，将它们规范化为 UTC（全球标准时间）或祖鲁时间。请注意，对于时间戳记，XML 模式数据类型允许的精度大于 SQL 数据类型允许的精度。如果遇到超出范围的值，那么将返回错误。

基于 XML 数据的索引的数据类型转换

在将值插入到XML 数据索引中之前，必须先将它们转换为与索引 SQL 数据类型对应的索引 XML 类型。

对于 VARCHAR(*integer*) 和 VARCHAR HASHED，通过使用 XQuery 函数 fn:string 将值转换为 xs:string 值。将 VARCHAR(*integer*) 的长度属性作为一种约束应用于生成的 xs:string 值。索引 SQL 数据类型 VARCHAR HASHED 将散列算法应用于生成的 xs:string 值，以生成插入到索引中的散列代码。将 VARCHAR 类型的数据直接存储在索引中，而不必先规范化为模式数据类型。

对于 DOUBLE、DATE 和 TIMESTAMP 索引，使用 XQuery 数据类型转换表达式将值转换为索引 XML 类型。在将 DATE 和 TIMESTAMP 数据类型值存储在索引中之前，将它们规范化为 UTC（全球标准时间）或祖鲁时间。如果某个 XML 数据根据 XQuery 规则有效，但由于系统局限性无法转换为索引数据类型，那么它将导致建立索引错误。索引 SQL 数据类型 DOUBLE 的值可能包括特殊数字值 *NaN*、*INF*、*-INF*、*+0* 和 *-0*，即使 SQL 数据类型 DOUBLE 本身不支持这些值。

相应的索引数据类型

表 23. 相应的索引数据类型

XML 数据类型	SQL 数据类型
xs:string	VARCHAR(*integer*) 和 VARCHAR HASHED
xs:double	DOUBLE
xs:date	DATE
xs:dateTime	TIMESTAMP

无效 XML 值

XML 模式值是 CREATE INDEX 语句的 *xmlpattern-clause* 生成的已建立索引的值。对于使用数据类型 DOUBLE、DATE 和 TIMESTAMP 的索引，使用 XQuery 数据类型转换表达式将 XML 模式值转换为索引 XML 数据类型。没有对目标索引 XML 数据类型形成有效的词法格式的 XML 值被视为无效 XML 值。

例如，ABC 是 xs:double 数据类型的无效 XML 值。索引处理无效 XML 值的方式取决于 CREATE INDEX 语句的 *xmltype-clause* 中指定的选项。

REJECT INVALID VALUES

如果指定了 REJECT INVALID VALUES 选项，那么所有 XML 模式值必须对索引 XML 数据类型有效。如果任何 XML 模式值不能转换为索引 XML 数据类型，那么会返回错误。如果索引已存在，那么不能在表中插入或更新 XML 数据。如果索引不存在，那么不会创建索引。

例如，假定用户创建索引 EMPID，它会将数字职员标识作为 DOUBLE 数据类型建立索引。将对 31201 之类的数字值建立索引。 但是，如果其中一个文档错误地将部门标识值 M55 作为其中一个职员标识属性值使用，那么文档插入会失败并产生错误消息，原因是 M55 是无效 DOUBLE 值。

```
CREATE INDEX EMPID ON DEPARTMENT(DEPTDOCS)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
   REJECT INVALID VALUES
```

IGNORE INVALID VALUES

如果指定了 IGNORE INVALID VALUES 选项，那么会忽略对目标索引 XML 数据类型无效的 XML 模式值。CREATE INDEX 语句不会为已存储 XML 文档中的对应值建立索引。这是缺省情况。在插入和更新操作期间，不会为无效 XML 模式值建立索引，但 XML 文档仍然会插入到表中。不会返回错误或警告

的原因是指定这些数据类型不会被视为对 XML 模式值的约束，但主要是因为搜索特定 XML 索引数据类型的 XQuery 表达式决不会考虑这些值。

当创建使用 DOUBLE、DATE 或 TIMESTAMP 数据类型的索引时，试图将值转换为目标索引 XML 类型。如果值对于目标索引 XML 数据类型（对于 xs:double、xs:date 或 xs:dateTime）是无效的 XML 数据，那么不会对该值建立索引，并且不会发出错误或警告。对于使用 VARCHAR 数据类型的索引，由于使用了 fn:string，因此所有值都使用 VARCHAR 数据类型，表示索引的有效 XML 数据。

请注意，索引只能忽略对于数据类型无效的 XML 值。有效值必须符合索引 XML 数据类型的值的 DB2 数据库服务器表示，否则会发出错误。与索引 XML 数据类型 xs:string 相关联的 XML 值一直有效。但是，如果超出最大长度，关联索引 SQL 数据类型 VARCHAR(*integer*) 数据类型的附加长度约束仍然会导致错误。如果返回错误，那么索引已存在时不会在表中插入或更新 XML 数据。如果索引不存在，那么不会创建索引。

忽略对数据类型无效的 XML 模式值时，目标索引 XML 数据类型就像过滤器，它不是约束，因为对于同一 XML 列，用户可以具有多个不同数据类型的索引。例如，假定用户对同一模式创建两个不同数据类型的索引。索引 ALLID 使用 VARCHAR 数据类型并对文档中的所有标识建立索引（部门标识和职员标识）。索引 EMPID 只对数字职员标识建立索引并使用 DOUBLE 数据类型作为过滤器:

使用显式 IGNORE INVALID VALUES 选项

```
CREATE INDEX ALLID ON DEPARTMENT(DEPTDOCS)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(10)
   IGNORE INVALID VALUES

CREATE INDEX EMPID ON DEPARTMENT(DEPTDOCS)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
   IGNORE INVALID VALUES
```

逻辑上等价的语句（使用缺省值）

```
CREATE INDEX ALLID ON DEPARTMENT(DEPTDOCS)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(10)

CREATE INDEX EMPID ON DEPARTMENT(DEPTDOCS)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
```

部门标识值 `M25` 是有效 VARCHAR 数据类型值，并且会插入到索引 ALLID 中。但是，`M25` 无法转换为 DOUBLE 数据类型，所以值将不能插入到 EMPID 中，并且不会产生错误或警告。将为存储在表中的文档插入值。

尽管 DOUBLE 索引 EMPID 中不存在值 `M25`，但查询仍可使用 DOUBLE 索引来检索所有匹配的数字值，并且将不会发生转换错误，因为将不访问包含 `M25` 的文档。

但是，如果查询不使用 DOUBLE 索引 EMPID 但使用 `//@id=25` 谓词来扫描文档，那么会发生转换错误，因为值 `M25` 与模式匹配并且仍存在于文档中，但它不是数字值。

请注意，文档中的所有值对于 xs:string（SQL VARCHAR）数据类型有效。仅对于数据类型 xs:double（SQL DOUBLE）、xs:date（SQL DATE）和 xs:dateTime（SQL TIMESTAMP），才会出现值不插入到索引中的情况。

文档遭拒或 CREATE INDEX 语句失败

对于下列类型的建立索引错误，INSERT 或 UPDATE 语句将拒绝 XML 文档（SQLSTATE 23525 和 sqlcode -20305）。如果对已经存在 XML 文档的已填充表执行 CREATE INDEX 语句，那么 CREATE INDEX 语句将失败（SQLSTATE 23526 和 sqlcode -20306），但该文档仍存储在表中。

VARCHAR(*integer*) 长度约束错误
一个或多个 XML 模式表达式生成的索引值的长度超出用户指定的 VARCHAR 数据类型的长度约束。

列表数据类型节点不受支持错误
XML 值中的一个或多个 XML 节点值是指定索引无法建立索引的列表数据类型节点。基于 XML 数据的索引不支持列表数据类型节点。

转换错误
如果源值对索引 XML 数据类型无效并且使用 CREATE INDEX 语句指定了 REJECT INVALID VALUES 选项，那么会返回错误。如果源值是不能转换为模式数据类型或索引 XML 数据类型的 DB2 数据库服务器表示的有效 XML 值（因为内部 DB2 局限性），那么会发出错误。必须发出该错误以维护一致性结果：如果执行了使用该索引的查询，那么查询的正确结果可能包含超出受支持限制的值（原因是该值是有效 XML 值）。为避免查询返回不完整的结果，会发出错误以维护一致的结果。

表 24. 一些内部 DB2 局限性示例

XML 数据类型	XML 模式	DB2 范围（最小值 ： 最大值）
xs:date	没有最大年份限制 支持日期为负数	0001-01-01: 9999-12-31
xs:dateTime	没有最大年份限制 支持日期为负数 支持任意精度的小数秒	0001-01-01T00:00:00.000000Z : 9999-12-31T23:59:59.999999Z
xs:integer	对于最小值或最大值范围没有限制	-9223372036854775808: 9223372036854775807

DB2 数据库服务器不支持 XML 值的完整范围。不支持的值范围包括:

- 年份大于 9999 或小于 0 的日期或日期时间值
- 小数秒精度大于 6 位的日期或日期时间值
- 数字值超出范围

关于转换为索引 XML 数据类型的总结表

要使数据成功转换为目标索引 XML 数据类型，根据模式数据类型和索引 XML 数据类型，源值在词法上必须有效，并且该值必须在模式数据类型和索引 XML 数据类型的 DB2 限制内。

下表总结了将值转换为索引 XML 数据类型时如何处理这些值。

表 25. 关于转换为索引 XML 数据类型的总结表

根据索引 XML 数据类型，值有效（所有值对于 xs:string 数据类型都有效）	**值在索引 XML 数据类型的 DB2 限制内**	**建立索引结果**
否	不适用	REJECT INVALID VALUES：错误 IGNORE INVALID VALUES（缺省值）：该值被忽略并且不会对其建立索引。
是	是	对值建立索引。
是	否	错误：值超出 DB2 限制。

XML 模式和索引键生成

应检查 XML 模式，以便可对其数据类型匹配 XML 模式数据类型规范的 XML 列创建索引。决定要为索引选择的 XML 模式时，还应该考虑想要运行的查询。

如果使用 XML 模式，那么将验证要存储在 XML 列中的 XML 文档的结构，以便针对 XML 模式约束 XML 文档中的元素和属性的数据类型。如果文档不符合模式规范，那么解析器将会拒绝该文档。例如，如果模式指定将属性约束为 DOUBLE 数据类型，但文档属性的值是 *ABC*，那么将拒绝该文档。如果未使用 XML 模式，那么解析器将不会验证文档数据，并认为文档数据是无类型数据。

例如，假定使用以下 XML 模式：

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="product" type="ProdType"/>

<xsd:simpleType name="ColorType">
    <xsd:restriction base="xsd:string">
        <xsd:maxLength value='20'/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="ProdType">
      <xsd:sequence>
        <xsd:element name="name"    type="xsd:string" />
        <xsd:element name="SKU"     type="xsd:string" />
        <xsd:element name="price"   type="xsd:integer" />
        <xsd:element name="comment" type="xsd:string" />
      </xsd:sequence>
    <xsd:attribute name="color"   type="ColorType" />
    <xsd:attribute name="weight"  type="xsd:integer" />
  </xsd:complexType>
</xsd:schema>
```

在查看需要发出的查询之后，您可能决定它们需要基于 price 和 color 的索引。分析查询有助于确定要包括在 CREATE INDEX 语句中的 XML 模式表达式。XML 模式指导要为索引选取的数据类型：如果 price 元素是整数，那么可以为索引 *priceindex* 选择数字数据类型 DOUBLE，如果 color 属性是字符串，那么可以为索引 *colorindex* 选择数据类型 VARCHAR。

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.PRODUCTDOCS')/product[price > 5.00]
   return $i/name
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.PRODUCTDOCS')/product[@color = 'pink']
   return $i/name
```

```
CREATE INDEX priceindex on company(productdocs)
      GENERATE KEY USING XMLPATTERN '/product/price' AS DOUBLE
CREATE INDEX colorindex on company(productdocs)
      GENERATE KEY USING XMLPATTERN '//@color' AS SQL VARCHAR(80)
```

模式还可以为字符串数据类型指定其他约束，如 *maxLength*，它显示在示例的 *ColorType* 下面，其中字符串限制为 20 个 Unicode 字符。由于 CREATE INDEX 语句指定以字节而不是字符为单位的 VARCHAR 长度，所以模式长度可以乘以因子 4 来计算存储索引中的模式所允许的最长字符串所需的最大字节数。在此示例中，4*20 = 80，因此为 *colorindex* 选择 VARCHAR(80)。

如果模式没有为字符串数据类型指定长度约束，并且文档中值的最大字符串长度未知，那么可以使用索引所使用的页面大小允许的最大长度。索引将存储不同长度的字符串，但由于仅存储每个字符串所需的实际字节数，所以指定比所需最大长度稍长的长度不会占用更多存储器。但是，在索引扫描期间，需要分配内存中较大的键缓冲区来处理最大键大小。有关指定 VARCHAR 数据类型的 XML 列的索引所允许的最大长度列表，请参阅 CREATE INDEX 语句。

如果 VARCHAR 数据类型的最大长度不够，无法对文档值建立索引，那么可以使用没有长度限制的 VARCHAR HASHED 数据类型。但是，只能将使用 VARCHAR HASHED 的索引用于相等查找，而不能用于范围扫描。请注意，包含的字符串比为 VARCHAR (*integer*) 指定的长度要长的文档将被拒绝。

XML 模式还可以指定缺省属性和元素值。如果 XML 文档中未指定相应的值并且验证了该文档，那么存储文档时将使用模式中的缺省值。将对这些缺省值以及原始输入文档中的其他值建立索引。如果未验证文档，那么不会将缺省值添加至文档，并且不会对它们建立索引。

UNIQUE 关键字语义

对非 XML 列的索引和 XML 列的索引使用相同的 UNIQUE 关键字，但该关键字分别具有不同的含义。

对于关系索引，CREATE INDEX 语句中的 UNIQUE 关键字强制表中的所有行之间的唯一性。对于基于 XML 数据的索引，UNIQUE 关键字强制其节点由 XML 模式限定的所有文档中单个 XML 列内的唯一性。插入单个文档可能导致多个值插入到唯一索引中；这些值在该文档以及同一 XML 列中的所有其他文档中必须唯一。还需要注意，插入一些文档可能不会导致任何值插入到索引中；不会对这些文档强制唯一性。

在将 XML 值转换为对索引指定的 SQL 数据类型之后，可以对索引的数据类型、节点的 XML 路径和节点的值强制唯一性。

指定 UNIQUE 关键字时一定要小心。因为转换为索引的指定数据类型可能会导致精度或范围丢失，或者不同的值可能散列为相同键值，所以在 XML 文档中看似唯一的多个值可能导致键重复错误。下列情况可能出现键重复错误:

- 指定了 VARCHAR HASHED 时，唯一字符串可能散列为相同散列代码并导致键重复错误。

- 对于数值，超出 DOUBLE 数据类型范围的精度或值丢失可能导致插入期间出现键重复错误。例如，将较大的整数和无限大的十进制值作为 DOUBLE 数据类型存储在索引中时可能会丢失精度。

如果指定了 VARCHAR(*integer*)，那么会将 XML 文档中的整个字符串存储在索引中，这样就不会出现键重复错误。此外，字符串的唯一性还遵循 XQuery 语义，在该语义中尾部空格很重要。因此，在 SQL 中重复但尾部空格不同的值在XML 数据索引中视为唯一值。

```
CREATE UNIQUE INDEX EMPINDEX ON company(companydocs)
    GENERATE KEY USING XMLPATTERN '/company/emp/name/last' AS SQL
    VARCHAR(100)
```

对于 UNIQUE 索引，XML 模式必须指定一个完整的路径，并且不能包含下列任何内容:

- descendant 轴
- descendant-or-self 轴
- /descendant-or-self::node()/（//）
- 用于 XML 名称测试的任何通配符
- 用于 XML 类型测试的 node() 或 processing instruction()

与为 XML 数据建立索引相关联的数据库对象

基于 XML 数据的逻辑索引和物理索引

创建 XML 数据索引时，将创建两个 B 型树索引：逻辑索引和物理索引。

逻辑索引包含 CREATE INDEX 语句所指定的 XML 模式信息。物理索引具有 DB2 生成的键列来支持逻辑索引，并包含已建立索引的文档值，它已转换为在 DB2INSTDEF 语句的 *xmltype-clause* 中指定的数据类型。

在逻辑级别使用 XML 数据索引（例如，使用 CREATE INDEX 和 DROP INDEX 语句）。DB2 以透明方式处理底层物理索引。请注意，返回索引元数据的任何应用程序编程接口都无法识别物理索引。

在 SYSCAT.INDEXES 目录视图中，逻辑索引具有在 CREATE INDEX 语句中指定的索引名和索引类型 *XVIL*。物理索引具有系统生成的名称和索引类型 *XVIP*。总是先创建逻辑索引并为它指定一个索引标识（IID）。随后创建物理索引并为它指定下一个连续索引标识。

以下示例说明了逻辑索引和物理索引之间的关系：考虑两个基于 XML 数据的索引：*EMPINDEX* 和 *IDINDEX*。对于 *EMPINDEX*，逻辑索引具有名称 *EMPINDEX*、索引标识 *3* 和索引类型 *XVIL*。相应的物理索引具有系统生成的名称 *SQL060414134408390*、索引标识 *4* 和索引类型 *XVIP*。

表 26. 逻辑索引与物理索引之间的关系

索引名（**INDNAME**）	索引标识（**IID**）	表名（**TABNAME**）	索引类型（**INDEXTYPE**）
SQL060414133259940	1	COMPANY	XRGN
SQL060414133300150	2	COMPANY	XPTH

表 26. 逻辑索引与物理索引之间的关系 (续)

索引名（INDNAME）	索引标识（IID）	表名（TABNAME）	索引类型（INDEXTYPE）
EMPINDEX	3	COMPANY	XVIL
SQL060414134408390	4	COMPANY	XVIP
IDINDEX	5	COMPANY	XVIL
SQL060414134408620	6	COMPANY	XVIP

目录视图

有关每个目录视图的更多信息，请参阅“相关参考”一节。

SYSCAT.INDEXES
每行表示一个索引，包括基于 XML 数据的逻辑索引和物理索引。

SYSCAT.INDEXXMLPATTERNS
每行表示一个基于 XML 数据的索引中的一个模式子句。

数据库分区环境中的索引

可从任何数据库分区发出 CREATE INDEX 和 ALTER INDEX 语句。

如果对在多个数据分区之间进行了分区的表创建索引，那么该索引也会在这些数据分区之间进行分区。

审计

XML 列的索引将现有索引对象类型用于审计。仅审计逻辑索引，不审计物理索引。

与 XML 列关联的其他数据库对象

有两个与 XML 列相关联的索引，即内部索引和系统生成的索引。这些索引表示为 SYSCAT.INDEXES 目录视图。此外，具有 XML 数据的数据分区表的索引分区表示为 SYSCAT.INDEXPARTITIONS 目录视图。

XML 路径索引和 XML 区域索引

每次创建 XML 列时，DB2 都会自动对 XML 列创建 XML 路径索引。DB2 还会为表中的所有 XML 列创建单个 XML 区域索引。

XML 路径索引记录存储在 XML 列中的 XML 文档内的所有唯一路径。

XML 区域索引捕获将 XML 文档内部分割为若干区域的方式，这些区域是页面内的节点集。由于区域是页面内的节点集，所以如果使用可以在页面内存储更多节点的较大的页大小，那么可以减少区域索引条目数并提高性能。

对于数据分区表，XML 区域索引始终是分区索引，而 XML 列路径索引始终是非分区索引。

SYSCAT.INDEXES 中 XML 区域索引的表空间标识和对象标识是逻辑值，原因是区域索引始终是分区索引，并且 SYSCAT.INDEXES 中的单个条目是逻辑代表。表空间标识和对象标识与它们的相关联分区表的表空间标识和对象标识相同。

与分区表中的数据分区相关联的每个索引分区在 SYSIBM.SYSINDEXPARTITIONS 目录表都有一个条目，此条目中包含有关该索引分区的信息和统计信息。可通过 SYSCAT.INDEXPARTITIONS 目录视图来访问此信息。非分区索引在此目录表中没有任何条目。

XML 路径索引和 XML 区域索引都记录在 SYSCAT.INDEXES 中。请注意，任何返回索引元数据的应用程序编程接口都无法识别这些索引。

这些与 XML 列相关联的内部索引不同于用户创建的基于 XML 数据的索引。要对存储在 XML 列中的 XML 数据建立索引，只能通过使用诸如 CREATE INDEX 和 DROP INDEX 语句来处理 XML 列的逻辑索引。

目录视图

有关这些目录视图的更多信息，请参阅“相关参考”一节。

SYSCAT.INDEXES
: 每行表示一个索引，包括 XML 路径索引和 XML 区域索引。XML 路径索引在 SYSCAT.INDEXES.INDEXTYPE 中显示为 *XPTH*，而 XML 区域索引在 SYSCAT.INDEXES.INDEXTYPE 中显示为 *XRGN*。

SYSCAT.INDEXPARTITIONS
: 每一行表示与分区表中的数据分区相关联的索引分区。

重新创建基于 XML 数据的索引

在下列情况下，将重新创建 XML 数据索引:

- 在执行 REORG INDEX 或 REORG INDEXES 命令期间。
- 在执行 REORG TABLE 命令期间。
- 发出指定了 REPLACE 选项的 IMPORT 命令时。
- 当查询、插入、删除或更新操作尝试访问表或索引并检测到索引对象标记为无效时。

请注意，与本机 XML 数据存储功能关联的所有索引都包含在与关系索引相同的表索引对象中。这包括任何 XML 路径索引、XML 区域索引和可能存在的基于 XML 数据的索引。不会只重新创建单个索引。如果必须重新创建索引，那么会一起重新创建索引对象中的所有索引。

CREATE INDEX

CREATE INDEX 语句可用来:

- 对 DB2 表定义索引。可以对 XML 数据或关系数据定义索引。
- 创建索引规范（就是用来向优化器指示数据源表具有索引的元数据）

调用

可以将此语句嵌入应用程序中，还可以通过使用动态 SQL 语句来发出该语句。它是一个可执行语句，仅当 DYNAMICRULES 运行行为对于程序包有效时才能动态编译该语句（SQLSTATE 42509）。

权限

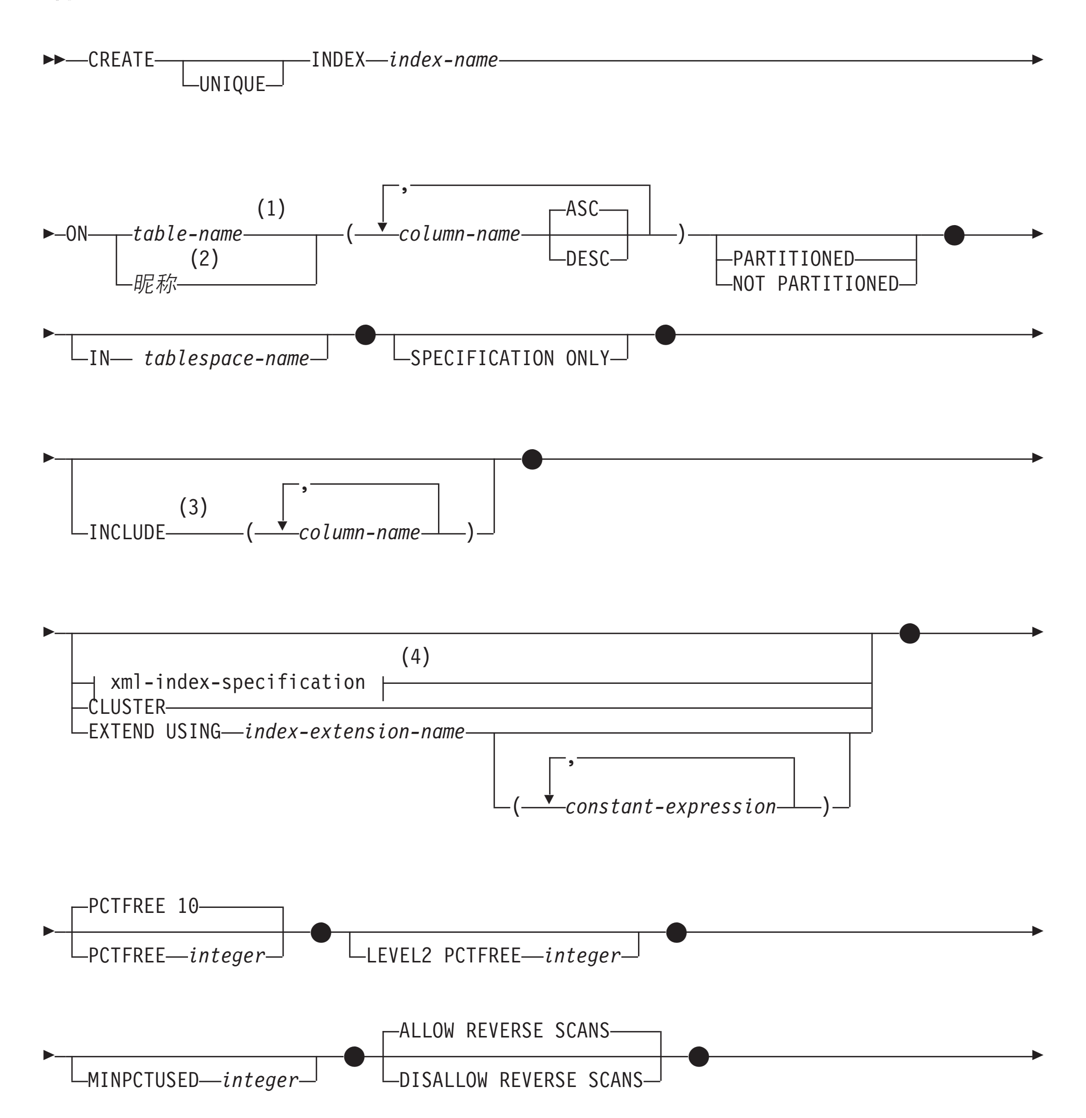
语句授权标识拥有的特权必须至少包括下列其中一项权限或特权:

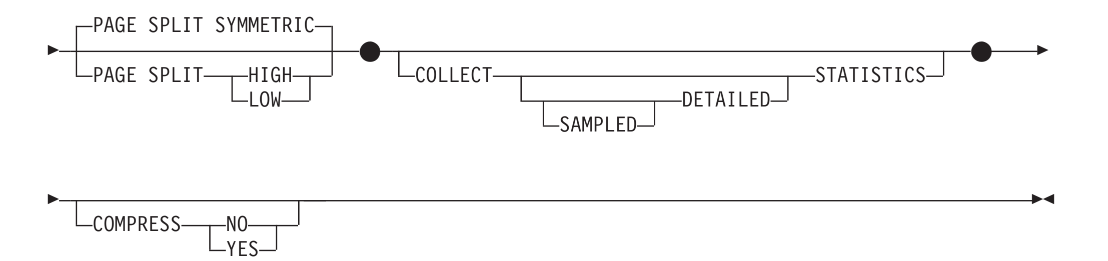
- 下列其中一项:
  - 对于定义了索引的表或昵称的 CONTROL 特权
  - 对于定义了索引的表或昵称的 INDEX 特权

  以及下列其中一项:
  - 对数据库的 IMPLICIT_SCHEMA 权限（如果该索引的隐式或显式模式名不存在）
  - 对模式的 CREATEIN 特权（如果该索引的模式名引用现有模式）
- DBADM 权限

对已声明临时表创建索引时不需要显式特权。

语法





注:

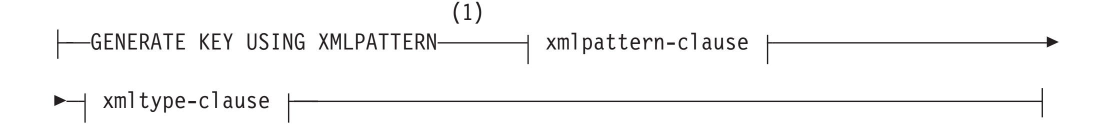
1　在联合系统中，*table-name* 必须标识联合数据库中的一个表。它不能标识数据源表。

2　如果指定了 *nickname*，那么 CREATE INDEX 语句将创建索引规范。在这种情况下，不能指定 INCLUDE、*xml-index-specification*、CLUSTER、EXTEND USING、PCTFREE、MINPCTUSED、DISALLOW REVERSE SCANS、ALLOW REVERSE SCANS、PAGE SPLIT 或 COLLECT STATISTICS。

3　仅当指定了 UNIQUE 时才能指定 INCLUDE 子句。

4　如果指定了 *xml-index-specification*，那么不能指定 *column-name* DESC、INCLUDE 或 CLUSTER。

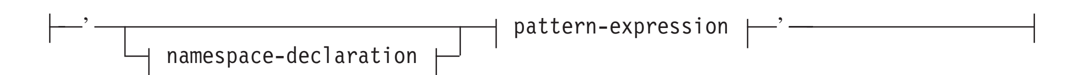
xml-index-specification:



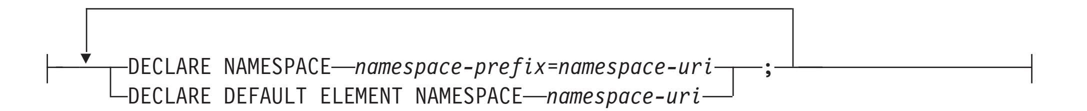
注:

1　可以使用备用语法 GENERATE KEYS USING XMLPATTERN。

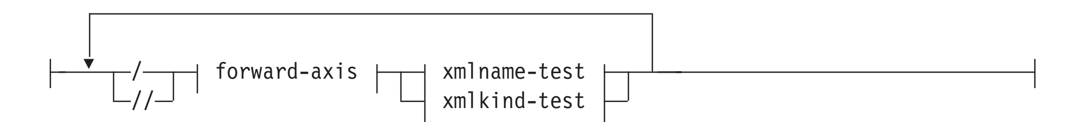
xmlpattern-clause:



namespace-declaration:



pattern-expression:



forward-axis:



xmlname-test:

```
├──┬─xml-qname──────────┬──────────────────────────────────────┤
   └─┤ xml-wildcard ├───┘
```

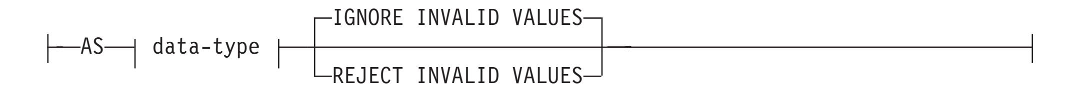
xml-wildcard:

```
├──┬─*───────────────┬──────────────────────────────────────────┤
   ├─xml-nsprefix:*──┤
   └─*:xml-ncname────┘
```

xmlkind-test:

```
├──┬─node()───────────────────────┬─────────────────────────────┤
   ├─text()───────────────────────┤
   ├─comment()────────────────────┤
   └─processing instruction()─────┘
```

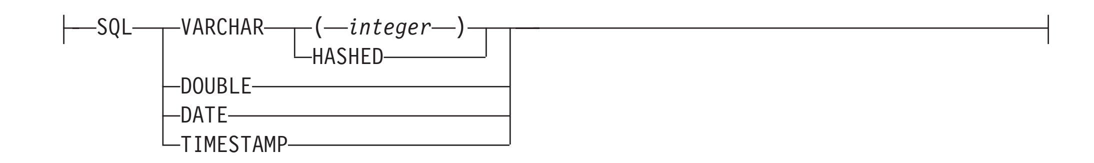
xmltype-clause:



data-type:

```
├──┤ sql-data-type ├────────────────────────────────────────────┤
```

sql-data-type:



描述

UNIQUE

如果指定了 ON *table-name*，那么 UNIQUE 将阻止表的两行或更多行具有相同的索引键值。在更新行或插入新行的 SQL 语句末尾将强制实施唯一性。

在执行 CREATE INDEX 语句期间也将检查唯一性。如果表中已经包含具有重复键值的行，那么不会创建索引。

如果是对 XML 列创建索引（索引是对 XML 数据创建的索引），那么唯一性适用于表中所有行的具有指定 *pattern-expression* 的值。对于已转换为指定 *sql-data-type* 的每个值都将强制实施唯一性。因为转换为指定的 *sql-data-type* 可能会导致精度或范围丢失，或者不同的值可能散列为相同键值，所以在 XML 文档中看似唯一的多个值可能导致键重复的错误。字符串的唯一性取决于 XQuery 语义，在该语义中，尾部空格很重要。因此，在 SQL 中重复但尾部空格不同的值在XML 数据索引中视为唯一值。

当使用 UNIQUE 时，会将空值当作任何其他值来对待。例如，如果键是可能包含空值的单个列，那么该列不能包含多个空值。

如果指定了 UNIQUE 选项，并且表具有分布键，那么索引键中的列必须是该分布键的超集。即，为唯一索引键指定的列必须包括分布键的所有列（SQLSTATE 42997）。

如果指定了 UNIQUE 选项并且表具有表分区键，那么索引键的列必须是该表分区键的超集。即，为唯一索引键指定的列必须包括该表分区键的所有列（SQLSTATE 42990）。

主键或唯一键不能是维的子集（SQLSTATE 429BE）。

如果指定了 ON *nickname*，那么仅当索引键的数据包含数据源表的每一行的唯一值时才应指定 UNIQUE。将不会检查唯一性。

对于 XML 数据索引，仅当指定的 *pattern-expression* 指定单个完整路径并且不包含 descendant 或 descendant-or-self AXIS、“//”、*xml-wildcard*、*node()* 或 *processing-instruction()* 时，才能指定 UNIQUE（SQLSTATE 429BS）。

在分区数据库环境中，带有一个或多个 XML 列的表存在以下规则:

- 分布式表不能有基于 XML 数据的唯一索引。
- 基于 XML 数据的唯一索引仅在没有分布键并且位于单节点多分区数据库上的表中受支持。
- 如果表存在基于 XML 数据的唯一索引，那么不能改变该表来添加分布键。

INDEX *index-name*

为索引或索引规范命名。该名称（包括隐式或显式限定符）一定不能标识在目录中所描述的索引或索引规范，也不能标识已声明临时表的现有索引（SQLSTATE 42704）。限定符不能是 SYSIBM、SYSCAT、SYSFUN 或 SYSSTAT（SQLSTATE 42939）。

已声明的全局临时表上索引的隐式或显式限定符必须为 SESSION（SQLSTATE 428EK）。

ON *table-name* **or** *nickname*

table-name 标识要对其创建索引的表。该表必须是一个基本表（而不是视图）、已创建的临时表、已声明的临时表、存在于当前服务器中的具体化查询表或者已声明的临时表。必须使用 SESSION 来限定已声明临时表的名称。*table-name* 不能标识目录表（SQLSTATE 42832）。如果指定了 UNIQUE 并且 *table-name* 是一个类型表，那么它不能是子表（SQLSTATE 429B3）。

nickname 是要对其创建索引规范的昵称。*nickname* 引用一个由索引规范描述其索引的数据源表，或者引用一个基于这样一个表的数据源视图。*nickname* 必须列示在目录中。

column-name

对于索引，*column-name* 标识要作为索引键的一部分的一列。而对于索引规范，*column-name* 是联合服务器用来引用数据源表的某列的名称。

每个 *column-name* 都必须是一个用来标识表中某列的非限定名称。最多可以指定 64 列。如果 *table-name* 是类型表，那么最多可以指定 63 列。如果 *table-name* 是一个子表，那么在该子表中必须至少引入一个 *column-name*；即，不是继承自一个超表（SQLSTATE 428DS）。不能重复指定 *column-name*（SQLSTATE 42711）。

指定列的已存储长度的总和不能大于页大小的索引键长度限制。要了解键长度限制，请参阅『SQL 限制』。如果 *table-name* 是类型表，那么索引键长度限制还要再缩短 4 个字节。注意，系统开销可能还会进一步减小此长度限制，将根据列的数据类型和列是否可空而有所变化。有关影响此限制的开销的更多信息，请参阅“CREATE TABLE”中的“字节计数”。

注意，系统开销可能会减小此长度，将根据列的数据类型和列是否可空而有所变化。有关影响此限制的开销的更多信息，请参阅“CREATE TABLE”中的“字节计数”。

不能将基于 LOB 的 LOB 列或单值类型列用作索引的一部分，即使该列的长度属性足够小，能够满足页大小的索引键长度限制也是如此（SQLSTATE 54008）。仅当还指定了 EXTEND USING 子句时，才能指定结构化类型列（SQLSTATE 42962）。如果指定了 EXTEND USING 子句，那么只能指定一列，并且该列的数据类型必须是不基于 LOB 的结构化类型或单值类型（SQLSTATE 42997）。

如果索引只有一列，其数据类型为 XML，并且还指定了 GENERATE KEY USING XMLPATTERN 子句，那么该索引是一个 XML 数据索引。仅当还指定了 GENERATE KEY USING XMLPATTERN 子句时，才能指定数据类型为 XML 的列（SQLSTATE 42962）。如果指定了 GENERATE KEY USING XMLPATTERN 子句，那么只能指定一列，并且该列的类型必须是 XML。

ASC

指定要按列值的升序来保存索引条目；这是缺省设置。对于使用 EXTEND USING 定义的索引不能指定 ASC（SQLSTATE 42601）。

DESC

指定要按列值的降序来保存索引条目。对于使用 EXTEND USING 定义的索引或者如果索引是 XML 数据索引，那么不能指定 DESC（SQLSTATE 42601）。

PARTITIONED

指示应创建分区索引。*table-name* 必须标识一个对数据分区定义的表（SQLSTATE 42601）。

如果该表是分区表，并且未指定 PARTITIONED 和 NOT PARTITIONED，那么会作为分区索引来创建索引（具有少量的异常）。如果存在下列任何情况，那么将创建非分区索引而不是创建分区索引:

- 指定了 UNIQUE 并且索引键不包含所有表分区键列。
- 创建了空间索引。
- 定义了基于 XML 数据的索引。

如果一个分区索引的定义与非分区索引的定义相同，也不会认为此分区索引是一个重复索引。有关更多详细信息，请参阅本主题中的第 152 页的『规则』这一节。

不能为下列索引指定 PARTITIONED 关键字:

- 非分区表的索引（SQLSTATE 42601）
- 所定义的基于 XML 数据的索引（SQLSTATE 42613）
- 索引键不包含所有表分区键列的唯一索引（SQLSTATE 42990）
- 空间索引（SQLSTATE 42997）

不能对具有已拆离的从属表（例如，MQT）的分区表创建分区索引（SQLSTATE 55019）。

对于分区索引的索引分区，应按下列规则来确定如何放置表空间:

- 如果被建立索引的表是使用 CREATE TABLE 语句的 partition-tablespace-options INDEX IN 子句创建的，那么将在该 INDEX IN 子句指定的表空间中创建索引分区。
- 如果用于创建被建立索引的表的 CREATE TABLE 语句未指定 partition-tablespace-options INDEX IN 子句，那么将在它建立索引的相应数据分区所在的同一表空间中创建索引分区分区索引。

CREATE INDEX 语句的 IN 子句不支持分区索引（SQLSTATE 42601）。对于分区索引，将忽略 CREATE TABLE 语句的 tablespace-clauses INDEX IN 子句。

NOT PARTITIONED

指示应该创建跨越为表定义的所有数据分区的非分区索引。*table-name* 必须标识一个对数据分区定义的表（SQLSTATE 42601）。

如果一个非分区索引的定义与分区索引的定义相同，也不会认为此非分区索引是一个重复索引。有关更多详细信息，请参阅本主题中的第 152 页的『规则』这一节。

对于非分区索引，应按下列规则来确定如何放置表空间:

- 如果您指定 CREATE INDEX 语句的 IN 子句，那么非分区索引将放入该 IN 子句所指定的表空间中。
- 如果未指定 CREATE INDEX 语句的 IN 子句，那么将按照下列规则来确定非分区索引的表空间放置:
  - 如果被建立索引的表是使用 CREATE TABLE 语句的 tablespace-clauses INDEX IN 子句创建的，那么非分区索引将放入该 INDEX IN 子句所指定的表空间中。
  - 如果被建立索引的表不是使用 CREATE TABLE 语句的 tablespace-clauses INDEX IN 子句创建的，那么将在该表的第一个可视或相连数据分区的表空间

中创建非分区索引。该表的第一个可视或相连数据分区就是按照范围规范排序的数据分区列表中的第一个分区。此外，该语句的授权标识不需要对缺省表空间具有 USE 特权。

IN *tablespace-name*

IN 子句仅支持非分区索引。对分区索引指定 IN 子句将导致 SQLSTATE 42601。

指定要在其中创建索引的表空间。此子句不支持对已创建的临时表或者已声明的临时表建立的索引（SQLSTATE 42601）。即使创建表时指定了 INDEX IN 子句，也可以指定此子句。这将覆盖前一子句。

由 *tablespace-name* 指定的表空间必须与表的数据表空间在同一个数据库分区组中，并且按照管理分区表的其他表空间的方式来管理空间；它必须是一个语句的授权标识具有 USE 特权的表空间（SQLSTATE 42838）。

如果未指定 IN 子句，那么会在由 CREATE TABLE 语句上的 INDEX IN 子句指定的表空间中创建索引。如果未指定 INDEX IN 子句，那么使用表的第一个可视或相连数据分区的表空间。这是按照范围规范排序的数据分区列表中的第一个分区。如果未指定 IN 子句，那么语句的授权标识不需要对缺省表空间具有 USE 特权。

SPECIFICATION ONLY

指示将使用此语句来创建应用于由 *nickname* 所引用的数据源表的索引规范。如果指定了 *nickname*，那么必须指定 SPECIFICATION ONLY（SQLSTATE 42601）。如果指定了 *table-name*，那么不能指定 SPECIFICATION ONLY（SQLSTATE 42601）。

如果将索引规范应用于唯一索引，那么 DB2 不会验证远程表中的列值是否是唯一的。如果远程列值不是唯一的，那么对包含索引列的昵称进行查询时可能会返回不正确的数据或者返回错误。

对已创建的临时表或者已声明的临时表创建索引时，不能使用此子句（SQLSTATE 42995）。

INCLUDE

此关键字引入一个子句，该子句指定要追加至一组索引键列的其他列。使用此子句所包含的任何列未用来强制唯一性。包含的这些列可能会通过仅访问索引来改进某些查询的性能。这些列必须与用来强制唯一性的列不同（SQLSTATE 42711）。当指定了 INCLUDE 时，必须指定 UNIQUE（SQLSTATE 42613）。对于列数和总长度属性的限制适用于唯一键和索引中的所有列。

不能对已创建的临时表或者已声明的临时表使用此子句（SQLSTATE 42995）。

column-name

标识包含在索引中但不是唯一索引键的一部分的一列。遵循与定义唯一索引键的列相同的规则。可以在 column-name 后面指定关键字 ASC 或 DESC，但是它们对顺序将不起作用。

如果指定了 *nickname*，或者索引是 XML 值索引，那么不能对使用 EXTEND USING 定义的索引指定 INCLUDE（SQLSTATE 42601）。

xml-index-specification

指定如何从存储在 XML 列中的 XML 文档来生成索引键。如果有多个索引列，或者如果列不具有 XML 数据类型，那么不能指定 *xml-index-specification*。

此子句仅适用于 XML 列（SQLSTATE 429BS）。

GENERATE KEY USING XMLPATTERN *xmlpattern-clause*

指定要对 XML 文档中的哪些部分建立索引。XML 模式值是 *xmlpattern-*

clause 生成的已建立索引的值。在索引中不支持列表数据类型节点。如果节点由 *xmlpattern-clause* 限定，并且存在一种指定节点是列表数据类型的 XML 模式，那么不能对该列表数据类型节点建立索引（对于 CREATE INDEX 语句将返回 SQLSTATE 23526，而对于 INSERT 和 UPDATE 语句将返回 SQLSTATE 23525）。

xmlpattern-clause

包含一个用来标识要建立索引的节点的模式表达式。它由一个可选的 *namespace-declaration* 和一个必需的 *pattern-expression* 组成。

namespace-declaration

如果模式表达式包含限定名，那么必须指定 *namespace-declaration* 以定义名称空间前缀。可以为非限定名称定义缺省名称空间。

DECLARE NAMESPACE *namespace-prefix***=***namespace-uri*

将 *namespace-prefix*（它是一个 NCName）映射至 *namespace-uri*（它是字符串文字）。*namespace-declaration* 可以包含多个 *namespace-prefix* 至 *namespace-uri* 的映射。*namespace-prefix* 在 *namespace-declaration* 列表中必须是唯一的（SQLSTATE 10503）。

DECLARE DEFAULT ELEMENT NAMESPACE *namespace-uri*

为非限定元素名称或类型声明缺省名称空间 URI。如果未声明缺省名称空间，那么元素和类型的非限定名称将不在任何名称空间中。只能声明一个缺省名称空间（SQLSTATE 10502）。

pattern-expression

指定 XML 文档中已建立索引的节点。*pattern-expression* 可以包含模式匹配字符（*）。它类似于 XQuery 中的路径表达式，但是它支持 DB2 所支持的 XQuery 语言的子集。

/（*正斜杠*）

将各个路径表达式步骤分隔开。

//（*双正斜杠*）

这是 /descendant-or-self::node()/ 的缩写语法。如果您还指定了 UNIQUE，那么不能使用 //（*双正斜杠*）。

forward-axis

child::

指定上下文节点的子代。如果未指定其他正向轴，那么这是缺省情况。

@ 指定上下文节点的属性。这是 attribute:: 的缩写语法。

attribute::

指定上下文节点的属性。

descendant::

指定上下文节点的后代。如果还指定了 UNIQUE，那么不能使用 *descendant::*。

self::

仅指定上下文节点本身。

descendant-or-self::
指定上下文节点及其后代。如果还指定了 UNIQUE，那么不能使用 descendant-or-self::。

xmlname-test
使用限定 XML 名称（xml-qname）或通配符（xml-wildcard）来指定路径中的步骤的节点名。

xml-ncname
按照 XML 1.0 定义的 XML 名称。该名称中不能包含冒号字符。

xml-qname
指定限定 XML 名称（也称为 QName），该名称可以具有两种格式:

- xml-nsprefix:xml-ncname，其中 xml-nsprefix 是一个用来标识名称空间作用域的 xml-ncname。
- xml-ncname，它指示应该将缺省名称空间作为隐式 xml-nsprefix 来应用。

xml-wildcard
将 xml-qname 指定为一个通配符，它可以具有以下三种格式:

- *（单个星号字符），它指示任何 xml-qname
- *xml-nsprefix*:*，它指示指定名称空间中的任何 xml-ncname
- *:*xml-ncname*，它指示任何名称空间作用域中的特定 XML 名称

如果还指定了 UNIQUE，那么不能使用 *xml-wildcard*。

xmlkind-test
使用这些选项来指定您的模式与哪些类型的节点相匹配。为您提供了下列选项:

node()
与任何节点相匹配。如果还指定了 UNIQUE，那么不能使用 *node()*。

text()
与任何文本节点相匹配。

comment()
与任何注释节点相匹配。

processing-instruction()
与任何处理指令节点相匹配。如果还指定了 UNIQUE，那么不能使用 processing-instruction()。

xmltype-clause

AS *data-type*
指定在存储已建立索引的值之前要将它们转换为的数据类型。这些值将被转换为与指定的索引 SQL 数据类型对应的索引 XML 数据类型。

表 27. 相应的索引数据类型

索引 XML 数据类型	索引 SQL 数据类型
xs:string	VARCHAR(*integer*)、VARCHAR 和 HASHED
xs:double	DOUBLE
xs:date	DATE
xs:dateTime	TIMESTAMP

对于 VARCHAR(*integer*) 和 VARCHAR HASHED，通过使用 XQuery 函数 fn:string 将值转换为 xs:string 值。将 VARCHAR(*integer*) 的长度属性作为一种约束应用于生成的 xs:string 值。索引 SQL 数据类型 VARCHAR HASHED 将散列算法应用于生成的 xs:string 值，以生成插入到索引中的散列代码。

对于使用数据类型 DOUBLE、DATE 和 TIMESTAMP 的索引，使用 XQuery 强制类型转换表达式将值转换为索引 XML 数据类型。

如果索引是唯一索引，那么将值转换为已建立索引的类型之后就会对它强制实行唯一性。

data-type

支持下列数据类型:

sql-data-type

受支持的 SQL 数据类型包括:

VARCHAR(*integer*)

如果指定了这种形式的 VARCHAR，那么 DB2 使用 *integer* 作为约束。如果要建立索引的文档节点的值长于 *integer*，那么如果索引已存在，就不会将文档插入表中。如果索引不存在，那么不会创建索引。*integer* 是介于 1 和依赖于页大小的最大值之间的一个值。表 28 显示每个页大小的最大值。

表 28. 使用不同页大小的文档节点的最大长度

页大小	文档节点的最大长度（按字节计）
4KB	817
8KB	1841
16KB	3889
32KB	7985

XQuery 语义用于字符串比较，其中的尾部空格非常重要。这与 SQL 语义不同，其中的尾部空格在比较期间不重要。

VARCHAR HASHED

指定 VARCHAR HASHED 以处理为任意长度的字符串建立索引。已建立索引的字符串的长度不受限制。DB2 将对整个字符串生成一个 8 字节的散列代码。只能将使用这些散列字符串的索引用于相等查询。XQuery 语义用于字符串相等比较，其中的尾部空格非常重要。这与 SQL 语义不同，其中的尾部空格在比较期间不重要。字符串散列保留 XQuery 相等语义，但不保留 SQL 语义。

DOUBLE

指定将数据类型 DOUBLE 用于为数字值建立索引。无限大的十进制类型和 64 位整数在作为 DOUBLE 值存储时可能会丢失精度。DOUBLE 的值可能包括特殊数字值 *NaN*、*INF*、*-INF*、*+0* 和 *-0*，即使 SQL 数据类型 DOUBLE 本身不支持这些值也是如此。

DATE

指定将数据类型 DATE 用于为 XML 值建立索引。注意，对于 xs:date，XML 模式数据类型允许值范围大于与 SQL 数据类型相对应的 DB2 pureXML xs:date 数据类型的值范围。如果遇到超出范围的值，那么将返回错误。

TIMESTAMP

指定将数据类型 TIMESTAMP 用于为 XML 值建立索引。注意，对于 xs:dateTime，XML 模式数据类型允许值和小数秒数精度的范围大于与 SQL 数据类型相对应的 DB2 pureXML xs:dateTime 数据类型的值和精度范围。如果遇到超出范围的值，那么将返回错误。

IGNORE INVALID VALUES

指定将忽略对目标索引 XML 数据类型无效的 XML 模式值，并且 CREATE INDEX 语句不会对已存储 XML 文档中的相应值建立索引。缺省情况下，将忽略无效值。在插入和更新操作期间，不会对无效 XML 模式值建立索引，但 XML 文档仍会插入表中。不会产生错误或警告，这是因为指定这些数据类型不会被视为对 XML 模式值的约束（搜索特定 XML 索引数据类型的 XQuery 表达式不考虑这些值）。

索引只能忽略对于索引 XML 数据类型无效的 XML 模式值。有效值必须符合索引 XML 数据类型的值的 DB2 表示，否则会返回错误。与索引 XML 数据类型 xs:string 关联的 XML 模式值始终有效。但是，如果超过最大长度，那么关联索引 SQL 数据类型 VARCHAR (*integer*) 的附加长度约束仍然会导致错误。如果返回错误，那么索引已存在时不能在表中插入或更新 XML 数据（SQLSTATE 23525）。如果索引不存在，那么不创建索引（SQLSTATE 23526）。

REJECT INVALID VALUES

指定所有 XML 模式值必须对索引 XML 数据类型有效。如果任何 XML 模式值不能强制转换为索引 XML 数据类型，那么会返回错误。如果索引已存在，那么不能在表中插入或更新 XML 数据（SQLSTATE 23525）。如果索引不存在，那么不创建索引（SQLSTATE 23526）。

CLUSTER

指定索引就是表的集群索引。将数据插入相关联的表时，通过尝试在此索引的键值属于同一范围内的那些行附近插入新行，来动态地维护或改进集群索引的集群因子。一个表只能存在一个集群索引。因此，如果在表的任何现有索引的定义中使用了 CLUSTER，就不能再指定 CLUSTER（SQLSTATE 55012）。不能对定义为使用追加方式的表创建集群索引（SQLSTATE 428D8）。

如果指定了 *nickname*，或者索引是 XML 数据索引，那么不允许指定 CLUSTER（SQLSTATE 42601）。不能对已创建的临时表或者已声明的临时表（SQLSTATE 42995）或者范围集群表（SQLSTATE 429BG）使用此子句。

EXTEND USING *index-extension-name*

对用来管理此索引的 *index-extension* 命名。如果指定了此子句，那么只能指定了一个 *column-name*，并且该列必须为结构化类型或单值类型（SQLSTATE 42997）。*index-extension-name* 必须是对目录中描述的索引扩展命名（SQLSTATE 42704）。对于单值类型，该列必须与索引扩展中相应源键参数的类型精确匹配。对于结构化类型列，相应的源键参数的类型必须与列类型相同或者是列类型的超类型（SQLSTATE 428E0）。

不能对已创建的临时表或者已声明的临时表使用此子句（SQLSTATE 42995）。

constant-expression

标识索引扩展的任何必需参数的值。每个表达式必须是一个常量值，该值的数据类型必须与已定义的相应索引扩展参数的数据类型（包括长度、精度和小数位在内）精确匹配（SQLSTATE 428E0）。在数据库代码页中，此子句的长度不能超过 32768 个字节（SQLSTATE 22001）。

PCTFREE *integer*

指定在构建索引时每个索引页中要保留的可用空间的百分比。将在页面中无限制地添加第一个条目。当将其他条目放入索引页时，在每页上至少要保留百分之 *integer* 的可用空间。*integer* 值的范围是 0 到 99。如果指定的值大于 10，那么在非叶子页中将只保留 10% 的可用空间。缺省值是 10。

如果指定了 *nickname*，那么不允许指定 PCTFREE（SQLSTATE 42601）。不能对已创建的临时表或者已声明的临时表使用此子句（SQLSTATE 42995）。

LEVEL2 PCTFREE *integer*

指定在构建索引时第 2 级的每个索引页中要保留的可用空间的百分比。*integer* 值的范围是 0 到 99。如果未设置 LEVEL2 PCTFREE，那么在所有非叶子页上至少要保留 10% 或者保留百分之 PCTFREE 的可用空间。如果设置了 LEVEL2 PCTFREE，那么在第 2 级中间页上保留了百分之 *integer* 的可用空间，而在第 3 级和更高级别的中间页上至少要保留 10% 或者保留百分之 *integer* 的可用空间。

如果指定了 *nickname*，那么不允许指定 LEVEL2 PCTFREE（SQLSTATE 42601）。不能对已创建的临时表或者已声明的临时表使用此子句（SQLSTATE 42995）。

MINPCTUSED *integer*

指示是否联机合并索引叶子页，以及在索引叶子页上使用的空间的最小百分比的阈值。从索引叶子页除去键之后，如果在页上使用的空间的百分比小于或等于百分之 *integer*，那么尝试将此页上的其余键与相邻页上的键进行合并。如果这些页上有足够的空间，那么会执行合并，并删除其中一页。*integer* 值的范围是 0 到 99。从性能方面考虑，建议您使用一个小于或等于 50 的值。指定此选项将影响更新和删除性能。当挂起了互斥表锁定时，将只在执行更新和删除操作期间进行合并。如果不存在互斥表锁定，那么在执行更新和删除操作期间，键将被标记为伪删除，并且不进行合并。考虑使用 REORG INDEXES 的 CLEANUP ONLY ALL 选项来合并叶子页，而不使用 CREATE INDEX 的 MINPCTUSED 选项。

如果指定了 *nickname*，那么不允许指定 MINPCTUSED（SQLSTATE 42601）。不能对已创建的临时表或者已声明的临时表使用此子句（SQLSTATE 42995）。

DISALLOW REVERSE SCANS

指定仅支持正向扫描索引或者按照在创建索引时定义的顺序来扫描索引。

不能同时指定 DISALLOW REVERSE SCANS 和 *nickname*（SQLSTATE 42601）。

ALLOW REVERSE SCANS

指定同时支持正向扫描和逆向扫描；即，按照在创建索引时定义的顺序来扫描索引，也可以按照相反顺序来扫描索引。

不能同时指定 ALLOW REVERSE SCANS 和 *nickname*（SQLSTATE 42601）。

PAGE SPLIT

指定索引分割行为。缺省值为 SYMMETRIC。

SYMMETRIC

指定在页的中间进行粗略分割。

HIGH

指定在插入索引键的值时高效率地使用索引页上的空间的索引页分割行为遵循特定模式。对于索引键值的子集，索引的最左边的一列或多列必须包含相同的值，而索引的最右边的一列或多列必须包含随着每次插入而增大的值。有关详细信息，请参阅"CREATE INDEX 语句的选项"。

LOW

指定在插入索引键的值时高效率地使用索引页上的空间的索引页分割行为遵循特定模式。对于索引键值的子集，索引的最左边的一列或多列必须包含相同的值，而索引的最右边的一列或多列必须包含随着每次插入而减小的值。有关详细信息，请参阅"CREATE INDEX 语句的选项"。

COLLECT STATISTICS

指定在创建索引期间要收集基本索引统计信息。

DETAILED

指定在创建索引期间还要收集扩充的索引统计信息（CLUSTERFACTOR 和 PAGE_FETCH_PAIRS）。

SAMPLED

指定在编译扩充的索引统计信息时可以使用采样。

COMPRESS

指定是否启用了索引压缩。缺省情况下，如果启用了数据行压缩，那么将启用索引压缩；如果禁用了数据行压缩，那么将禁用索引压缩。可以使用此选项来覆盖缺省行为。如果指定了 *nickname*，那么不允许指定 COMPRESS（SQLSTATE 42601）。

YES

指定启用了索引压缩。将根据压缩情况对索引执行插入和更新操作。

NO

指定禁用了索引压缩。

规则

- 尝试创建与现有索引相匹配的索引时，CREATE INDEX 语句将失败（SQLSTATE 01550）。

  有许多因素用来确定两个索引是否相匹配。这些因素按照各种不同的方式组合成一些规则来确定两个索引是否相匹配。使用下列因素来确定两个索引是否相匹配:

  1. 在两个索引中，索引列集合（包括任何 INCLUDE 列）相同。
  2. 在两个索引中，索引键列（包括任何 INCLUDE 列）的排序相同。
  3. 新索引的键列与现有索引中的键列相同或者是后者的超集。
  4. 在两个索引中，列的排序属性相同。

5. 现有索引是唯一的。
6. 两个索引都不是唯一的。

这些因素的下列组合构成了用于确定何时认为这两个索引相同的规则:

– 第 152 页的 1 + 第 152 页的2 + 第 152 页的4 + 5
– 第 152 页的 1 + 第 152 页的2 + 第 152 页的4 + 6
– 第 152 页的 1 + 第 152 页的2 + 第 152 页的3 + 5

异常:

– 如果进行比较的其中一个索引是分区索引，而进行比较的另一个索引是非分区索引，并且这两个索引的名称不同，那么认为它们不是重复索引，即使满足其他匹配索引条件也是如此。
– 对于 XML 数据的索引，如果索引名不同，那么即使已建立索引的 XML 列、XML 模式和数据类型（包括它的选项）都完全相同，也不认为索引描述是重复的。

- 系统维护的 MQT 的唯一索引不受支持（SQLSTATE 42809）。
- 如果指定了 nickname，那么不支持 COLLECT STATISTICS 选项（SQLSTATE 42601）。

注意

- 允许在创建索引时对表进行并发读/写访问。但是，对非分区表建立的索引、非分区索引和分区索引，缺省索引创建行为不同:
  – 对非分区表建立的索引，一旦构建了索引，就会将在创建索引期间对表所作的更改反映到新索引中。在创建索引时，对表的写访问会暂时被阻塞。创建索引之后，新索引就会变得可用。
  – 对于非分区索引，一旦构建了索引，就会将在创建索引期间对表所作的更改反映到新索引中。在创建索引时，对表的写访问会暂时被阻塞。创建索引之后，新索引就会变得可用。
  – 对于分区索引，一旦构建了索引分区，就会将在创建该索引分区期间对该分区所作的更改反映到新的索引分区中。在完成对其余数据分区创建索引时，将阻塞对该数据分区的写访问。在构建了最后一个数据分区的索引分区并且落实了事务之后，所有数据分区都可供读写。

  为了防止发生此缺省行为，可在发出 CREATE INDEX 语句之前使用 LOCK TABLE 语句来显式锁定表。（根据是否允许读访问，可以采用 SHARE 或 EXCLUSIVE 方式来锁定表。）
- 如果指定的表已包含数据，那么 CREATE INDEX 将为它创建索引条目。如果该表中尚不包含数据，那么 CREATE INDEX 将为索引创建描述；将数据插入表中时就会创建索引条目。
- 建议您一旦创建了索引并且将数据装入到表中，就发出 RUNSTATS 命令。RUNSTATS 命令将更新所搜集的有关数据库表、列和索引的统计信息。这些统计信息用来确定表的最佳访问路径。通过发出 RUNSTATS 命令，数据库管理器可以确定新索引的特征。如果在发出 CREATE INDEX 语句之前已经装入了数据，那么建议使用 CREATE INDEX 语句上的 COLLECT STATISTICS 选项来替代 RUNSTATS 命令。
- 使用一个尚不存在的模式名来创建索引时，如果语句的授权标识具有 IMPLICIT_SCHEMA 权限，那么会导致隐式创建该模式。模式所有者是 SYSIBM。将把对模式的 CREATEIN 特权授予给所有人（PUBLIC）。

- 在创建实际的索引之前，优化器可能会建议索引。
- 如果正在对具有索引的数据源表定义索引规范，那么索引规范的名称不必与索引名称相匹配。
- 优化器使用索引规范来改进对规范适用的数据源表的访问。
- *兼容性*: 为了与 DB2 z/OS® 版兼容:
  – 容许但是将忽略下列语法:
    - CLOSE
    - DEFINE
    - FREEPAGE
    - GBPCACHE
    - PIECESIZE
    - TYPE 2
    - using-block
  – 接受将下列语法作为缺省行为:
    - COPY NO
    - DEFER NO

示例

示例 1: 对 PROJECT 表创建一个名为 UNIQUE_NAM 的索引。创建索引的目的是为了确保表中条目的项目名（PROJNAME）不相同。索引条目将按升序排列。

```
CREATE UNIQUE INDEX UNIQUE_NAM
   ON PROJECT(PROJNAME)
```

示例 2: 对 EMPLOYEE 表创建一个名为 JOB_BY_DPT 的索引。按照每个部门（WORKDEPT）中职员的职称（JOB）来按升序排列索引条目。

```
CREATE INDEX JOB_BY_DPT
   ON EMPLOYEE (WORKDEPT, JOB)
```

示例 3: 昵称 EMPLOYEE 引用称为 CURRENT_EMP 的数据源表。在创建此昵称之后，就对 CURRENT_EMP 定义了索引。为索引键选择的列是 WORKDEBT 和 JOB。创建一个描述此索引的索引规范。通过此规范，优化器将知道索引存在及其索引键的内容。借助此信息，优化器就可以改进它访问表的策略。

```
CREATE UNIQUE INDEX JOB_BY_DEPT
  ON EMPLOYEE (WORKDEPT, JOB)
  SPECIFICATION ONLY
```

示例 4: 对结构化类型列位置创建一个名为 SPATIAL_INDEX 的扩展索引类型。索引扩展 GRID_EXTENSION 中的描述用来维护 SPATIAL_INDEX。为 GRID_EXTENSION 给定字面值以创建索引网格大小。

```
CREATE INDEX SPATIAL_INDEX ON CUSTOMER (LOCATION)
  EXTEND USING (GRID_EXTENSION (x'000100100010001000400010'))
```

示例 5: 对名为 TAB1 的表创建名为 IDX1 的索引，并收集有关 IDX1 索引的基本索引统计信息。

```
CREATE INDEX IDX1 ON TAB1 (col1) COLLECT STATISTICS
```

示例 6: 对名为 TAB1 的表创建名为 IDX2 的索引，并收集有关 IDX2 索引的详细索引统计信息。

```
CREATE INDEX IDX2 ON TAB1 (col2) COLLECT DETAILED STATISTICS
```

示例 7: 对名为 TAB1 的表创建名为 IDX3 的索引，并通过使用采样来收集有关 IDX3 索引的详细索引统计信息。

```
CREATE INDEX IDX3 ON TAB1 (col3) COLLECT SAMPLED DETAILED STATISTICS
```

示例 8: 对 IDX_TBSP 表空间中名为 MYNUMBERDATA 的分区表创建名为 A_IDX 的唯一索引。

```
CREATE UNIQUE INDEX A_IDX ON MYNUMBERDATA (A) IN IDX_TBSP
```

示例 9: 对 IDX_TBSP 表空间中名为 MYNUMBERDATA 的分区表创建名为 B_IDX 的非唯一索引。

```
CREATE INDEX B_IDX ON MYNUMBERDATA (B)
  NOT PARTITIONED IN IDX_TBSP
```

示例 10: 对名为 COMPANYINFO 的表创建一个 XML 数据索引，该 COMPANYINFO 表中包含一个名为 COMPANYDOCS 的 XML 列。XML 列 COMPANYDOCS 中包含与以下内容相似的大量 XML 文档:

```
<company name="Company1">
        <emp id="31201" salary="60000" gender="Female">
     <name>             <first>Laura</first>
            <last>Brown</last>
    </name>
        <dept id="M25">
        Finance
     </dept>
        </emp>
</company>
```

COMPANYINFO 表的用户通常需要使用职员标识来检索职员信息。与以下内容相似的索引可以使检索效率更高。

```
CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)
  GENERATE KEY USING XMLPATTERN '/company/emp/@id'
    AS SQL DOUBLE
```

示例 11: 以下索引与前一示例中创建的索引在逻辑上是等价的，只不过以下索引使用非缩写语法。

```
CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)
  GENERATE KEY USING XMLPATTERN '/child::company/child::emp/attribute::id'
    AS SQL DOUBLE
```

示例 12: 对名为 DOC 的列创建索引，仅将书名的索引创建为 VARCHAR(100)。因为书名对于所有书籍都应该是唯一的，所以该索引也必须是唯一的。

```
CREATE UNIQUE INDEX MYDOCSIDX ON MYDOCS(DOC)
  GENERATE KEY USING XMLPATTERN '/book/title'
    AS SQL VARCHAR(100)
```

示例 13: 对名为 DOC 的列创建索引，并将章节号的索引创建为 DOUBLE。此示例包含名称空间声明。

```
CREATE INDEX MYDOCSIDX ON MYDOCS(DOC)
  GENERATE KEY USING XMLPATTERN
           'declare namespace b="http://www.foobar.com/book/";
               declare namespace c="http://acme.org/chapters";
        /b:book/c:chapter/@number'
    AS SQL DOUBLE
```

针对基于 XML 数据的索引的样本查询

基于 XML 数据的索引需要与要利用它们的查询匹配。以下示例显示能够或不能利用基于 XML 数据的索引的查询。

可以使用 XML 数据索引的样本查询

使用多个不同谓词的查询可以利用 XML 数据索引。下面显示了与可以使用的索引匹配的 XQuery 谓词的一些示例。在匹配索引之后进行查询。

示例 1. 发出相等查询: 查找标识为 *42366* 的职员:

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@id='42366']
       return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(5)
```

示例 2. 范围查询: 查找薪水高于 *35000* 的职员:

```
XQUERY
   for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@salary > 35000]
   return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '//@salary' AS SQL DOUBLE
```

示例 3. 发出包含"或"（OR）的查询: 查找财务部或市场部的职员:

```
XQUERY
   for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[dept/text()='Finance'
   or dept/text()='Marketing']
   return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/dept/text()' AS SQL
      VARCHAR(30)
```

示例 4. 同一个索引可以满足不同查询:

查找标识为 *31201* 的职员:

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@id='31201']
        return $i
```

查找标识为 *K55* 的部门:

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp/dept[@id='K55']
        return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(25)
```

示例 5. 查询谓词可以包含路径: 查找销售部中姓 *Murphy* 的职员:

```
XQUERY
   for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[name/last='Murphy'
     and dept/text()='Sales']
   return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/name/last' AS SQL
      VARCHAR(100)
```

```
CREATE INDEX deptindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/dept/text()' AS SQL
      VARCHAR(30)
```

示例 6. 查询期间练习分层包含：查询可以使用索引在文档层次结构的不同级别执行"与"（AND）运算。查询还可以使用索引来确定哪些子节点属于同一祖代，以进行相应的过滤。

查找薪水等于 *60000* 的女职员所在的公司。在"有关为 XML 数据建立索引的概述"主题的样本 XML 片段中，Company1 和 Company2 都符合条件。

```
XQUERY for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company[emp/@salary=60000 and
   emp/@gender='Female']
   return $i
```

查找薪水等于 *60000* 的女职员。仅 Company1 中的 Laura Brown 符合条件。

```
XQUERY for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@salary=60000
   and @gender='Female']
   return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@salary' AS DOUBLE
```

```
CREATE INDEX genderindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@gender' AS SQL
      VARCHAR(10)
```

示例 7. 查询可以使用 descendant-or-self 轴（//）并使用索引，但前提是查询谓词的限制大于或者至少等于索引模式的限制。

查找部门标识为 *K55* 的职员:

```
XQUERY
   for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company//emp[.//dept//@id='K55' ]
   return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '//emp//@id' AS SQL VARCHAR(25)
```

示例 8. 对于多维集群（MDC）表，查询可使用 MDC 块索引和基于 XML 数据的索引。假定已创建带有 XML 列的 MDC 表，并且该表将 WORKDEPT 用作维。

```
CREATE TABLE employee (empno char(6), workdept char(3), doc xml)
ORGANIZE BY (workdept)
```

假定就像使用以下 CREATE INDEX 语句定义的那样，列 DOC 具有基于 XML 数据的索引:

```
CREATE INDEX hdate on employee(doc)
      GENERATE KEY USING XMLPATTERN '//hirdate'AS SQL DATE COLLECT STATISTICS
```

以下查询可在其访问方案中使用 WORKDEPT 的块索引和 DOC 的基于 XML 数据（hdate）的索引:

```
SELECT COUNT (*) FROM y.employee y
      WHERE workdept='A00'
      AND XMLEXISTS('$p/employee[hiredata > "1964-01-01"]
      PASSING y.doc as "p")
```

不能使用 XML 数据索引的样本查询

在某些情况下查询不能使用 XML 数据索引。下面列示了不能使用所显示的预期索引的 XQuery 谓词的一些示例。

示例 1. 在查询使用索引之前，查询所请求的数据类型必须与预期的数据类型匹配。在此示例中，查询请求字符串形式的职员标识，但建立的标识索引为数字:

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@id='31664']
        return $i

CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

示例 2. 用于创建索引的 XML 模式表达式的限制可能比查询谓词的限制更多。在此示例中，查询不能使用索引，因为查询同时检索部门标识和职员标识，但索引只包含职员标识:

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')//@id
        return $i

CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(5)
```

以下查询检索职员标识为 *31201* 或部门标识为 *K55* 的职员。由于标识可能是职员标识或部门标识，但索引只包含部门标识，所以不能按创建的那样使用索引。

```
XQUERY
   for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')//emp[.//@id='31201' or .//@id='K55']
   return $i

CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '//dept//@id' AS SQL VARCHAR(5)
```

对基于 XML 数据的索引的限制

建立基于 XML 数据的索引存在一些限制，包括数据类型支持限制、并行级别限制、XML 列表元素限制和索引压缩限制。

数据类型支持

在 CREATE INDEX 语句中指定的每个 XML 模式表达式必须与数据类型关联。仅支持四种基于 SQL 的数据类型：DATE、TIMESTAMP、VARCHAR 和 DOUBLE。

并行级别

处理 XML 列和相关联的索引时，将限制对一些并行级别的支持。下表描述了受支持或不受支持的并行级别:

表 29. 带有至少一个 XML 列的非分区表的受支持并行级别

函数	并行级别	是否受支持
REORG INDEXES ALL FOR TABLE	对表的并行读写访问权	是。

表 29. 带有至少一个 XML 列的非分区表的受支持并行级别 (续)

函数	并行级别	是否受支持
REORG TABLE	表子句：ALLOW [READ | NO] ACCESS	是。基于 XML 数据的索引可能存在。
REORG TABLE INPLACE（表至少存在一个基于 XML 数据的索引）	表子句：ALLOW [READ | WRITE] ACCESS	否。

表 30. 带有非分区索引以及至少一个 XML 列的分区表的受支持并行级别

函数	并行级别	是否受支持
REORG INDEX（对于基于 XML 数据的非分区索引）	对表的并行读写访问权	是。
REORG INDEXES ALL FOR TABLE	索引子句：ALLOW NO ACCESS 或 ALLOW [READ | WRITE] CLEANUP ONLY	是。
REORG INDEXES ALL FOR TABLE	索引子句：ALLOW [READ | WRITE] ACCESS	否。
REORG TABLE	表子句：ALLOW NO ACCESS	是。 **注**：对于分区表，唯一受支持的访问方式是 ALLOW NO ACCESS。
REORG TABLE INPLACE	表子句：ALLOW [NO | READ | WRITE] ACCESS	否。 **注**：对于分区表，不支持 INPLACE 选项。

有关子句和选项的信息，请参阅 CREATE INDEX 语句和 REORG INDEX/TABLE 命令。

对于多维集群（MDC）表，不支持 ALLOW WRITE ACCESS 方式的联机索引重新组织。

XML 列表元素

不能对列表数据类型节点建立索引。如果节点由 *xmlpattern-clause* 限定，并且存在一个指定节点是列表数据类型的 XML 模式，那么不能对该列表数据类型节点建立索引。对列表数据类型节点发出 CREATE INDEX 语句将返回错误（SQLSTATE 23526，sqlcode -20306）。发出 INSERT 和 UPDATE 语句还将返回错误（SQLSTATE 23525，sqlcode -20305）。

UNIQUE XML 数据索引

在分区数据库环境中，带有一个或多个 XML 列的表存在以下规则：

- 带有分布键的表不能有基于 XML 数据的唯一索引。
- 仅支持在单分区数据库上对没有分布键的表使用基于 XML 数据的唯一索引。
- 如果表存在基于 XML 数据的唯一索引，那么不能改变该表来添加分布键。

对 XML 列创建索引也要遵守对本机 XML 数据存储器设定的所有限制。请参阅下面的“相关参考”一节。

常见 XML 建立索引问题

如果在为 XML 数据建立索引时遇到问题，下列其中一个问题解决方案可能适用。

SQL20305N 和 SQL20306N 错误消息的问题确定

这些错误消息是在不能对 XML 节点值建立索引时发出的。SQL20305N 消息由 INSERT 和 UPDATE 语句以及 IMPORT 和 LOAD 实用程序发出。SQL20306N 消息由 CREATE INDEX 语句针对填充的基本表发出。

这些消息会输出错误的原因码。在命令行处理器中发出 `? SQL20305` 或 `? SQL20306`，以查找相应原因码的说明和用户响应。生成的 XQuery 语句会输出至 db2diag 日志文件，以便帮助您找出失败的 XML 节点值。

如果 SQL20305N 由 LOAD 实用程序发出，那么不会将生成的 XQuery 语句（用于找出失败节点值）写至 db2diag 日志文件。要生成这些 XQuery 语句，必须对未装入的失败行运行 IMPORT 实用程序。请参阅 DB2 信息中心内的"装入 XML 数据"和"解决装入 XML 数据时发生的建立索引错误"，以了解其他消息。

如果 SQL20305N 由 INSERT 或 UPDATE 语句发出，请参阅"诊断 INSERT 或 UPDATE 语句发出的 SQL20305N 消"。如果 SQL20306N 由 CREATE INDEX 语句发出，请参阅"诊断 CREATE INDEX 语句对已填充的表发出的 SQL20306N 消息"。

诊断 INSERT 或 UPDATE 语句发出的 SQL20305N 消息

要确定出现 SQL20305N 错误消息的原因，请参阅中"SQL20305N 和 SQL20306N 错误消息的问题确定"，然后遵循下列步骤:

1. 确定索引名和索引 XML 模式子句
   a. 通过使用错误消息中的 *index-id* 发出以下查询，以获取索引名（*index-name*,*index-schema*）:
   ```
   SELECT INDNAME,INDSCHEMA
      FROM SYSCAT.INDEXES
      WHERE IID = index-id AND
      TABSCHEMA ='schema' AND TABNAME ='table-name'
   ```
   b. 通过发出以下查询，使用 *index-name* 和 *index-schema* 来获取 SYSCAT.INDEXES 中的索引数据类型和 XML 模式:
   ```
   SELECT DATATYPE, PATTERN
      FROM SYSCAT.INDEXXMLPATTERNS
      WHERE INDSCHEMA = 'index-schema' AND
      INDNAME = 'index-name'
   ```
2. 要在输入文档中查找失败的节点值，请在 db2diag 日志文件中搜索字符串 `SQL20305N` 并与原因码编号匹配。依据原因码，您会找到一组指示信息和生成的 XQuery 语句，可使用它们来找出文档中导致该错误的值。对于较小的节点值，请在 XQuery 谓词中使用完整值。对于节点值太长而无法输出至 db2diag 日志文件的情况，将该值的起始字节与 XQuery 谓词中的 fn:starts-with 函数一起使用，并将该值的结束字节与 fn:ends-with 函数一起使用。
3. 因为文档被拒绝并且不在表中，所以不能对其运行 XQuery 语句。为解决此问题，请创建包含原始表中各列的新表，并在新表中插入失败文档。不要对新表创建任何索引。

4. 复制从 db2diag 日志文件中生成的 XQuery 语句，并将 XQuery 中的表名替换为新创建的表名。
5. 执行 XQuery 语句以检索完整文档和包含导致故障的值的文档片段。为了提供文档中出现错误的位置的上下文信息，XQuery 语句将输出文档片段，并以导致故障的节点值的父代开头。
6. 使用索引 XML 模式来标识要检查的匹配 XML 节点。因为生成的 XQuery 语句对名称空间使用通配符，所以要限定的多个问题值可能具有不同名称空间（但不常见）。如果发生这种情况，那么必须在索引 XML 模式中使用名称空间声明来确定匹配 XML 节点的正确集合。如果未在谓词中使用完整值来过滤结果，那么必须使用索引 XML 模式来验证 XQuery 语句返回的限定问题值。
7. 一旦在文档中找到失败的值，则应修改输入文档来更正该问题，并重新提交 INSERT 或 UPDATE 语句。

示例: INSERT 语句错误

在以下示例中，`hello world` 是无效 DOUBLE 值，而在生成的 XQuery 谓词中使用完整值。注意，`*N` 用作错误消息中模式信息不适用的位置的占位符。

```
CREATE TABLE t1 (x XML);

CREATE INDEX ix1 ON t1(x)
   GENERATE KEY USING XMLPATTERN '/root/x/text()'
   AS SQL DOUBLE REJECT INVALID VALUES;

DB20000I 已成功完成 SQL 命令。

INSERT INTO t1 VALUES (XMLPARSE (DOCUMENT
   'The beginning of the documenthello world'
   STRIP WHITESPACE));

DB21034E  该命令被当作 SQL 语句来处理，因为它是无效的"命令行处理器"命令。在 SQL 处理期间，它返回:

SQL20305N  由于在插入或更新表"ADUA.T"中的"IID = 23"所
标识的索引时检测到错误，无法插入或更新 XML 值。原因码 ="5"。
对于 XML 模式的相关原因码，XML 模式标识为"*N"，XML 模式数据类型为"*N"。
SQLSTATE=23525
```

db2diag 日志文件中的输出如下所示（包含少许格式更改）:

```
2007-03-06-12.02.08.116046-480 I4436A1141         级别: 警告
PID     : 1544348              TID  : 1801        PROC : db2sysc
INSTANCE: adua                 NODE : 000         DB   : ADTEST
APPHDL  : 0-18                 APPID: *LOCAL.adua.070306200203
AUTHID  : ADUA
EDUID   : 1801                 EDUNAME: db2agent (ADTEST)
函数: DB2 UDB, XML 存储器和索引管理器,
          xmlsIkaProcessErrorMsg, 探测: 651
消息: ZRC=0x80A50411=-2136669167=XMS_XML_IX_INSERT_UPDATE_ERROR
          "插入或更新 XML 索引时发生 XML 节点值错误"
DATA #1: 字符串, 36 字节
SQL 代码: SQL20305N; 原因码: 5
DATA #2: 字符串, 321 字节
要在文档中查找导致错误的值，请创建包含原始表中各列的新表，并在该表中插入失败文档。
不要对新表创建任何索引。在下面的查询中将表名替换为新创建的表名并执行以下 XQuery。
DATA #3: 字符串, 187 字节
xquery for $i in db2-fn:xmlcolumn("ADUA.T.X")[/*:root/*:x/text()="hello world"]
return
```

```
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:root/*:x/text()/..} </ProblemValue>
</Result>;
```

要查询失败节点值，请执行下列操作:

1. 创建包含原始表中各列的新表:

   ```
   CREATE TABLE t2 LIKE t1;
   ```

2. 将失败文档插入到新表中:

   ```
   INSERT INTO t2 VALUES (XMLPARSE (DOCUMENT
   'The beginning of the documenthello world'
   STRIP WHITESPACE));
   ```

3. 复制从 db2diag 日志文件中生成的 XQuery 语句，并将 XQuery 中的表名替换为新表名:

   ```
   xquery for $i in db2-fn:xmlcolumn("ADUA.T2.X")[/*:root/*:x/text()="hello world"]
   return
   {$i}
   {$i/*:root/*:x/text()/..}
   ;
   ```

4. 对新表执行 XQuery 语句。查询语句的结果如下所示（包含少许格式更改）:

   ```
   <Result>
      <ProblemDocument>
         <root>The beginning of the document<x>hello world</x></root>
      </ProblemDocument>
      <ProblemValue><x>hello world</x></ProblemValue>
   </Result>
   ```

更正以下错误:

可更改文档以使 <x> 元素包含将成功地转换为 DOUBLE 数据类型的数字值:

```
INSERT INTO t1 VALUES (
   XMLPARSE (DOCUMENT
   '<root>The beginning of the document<x>123</x></root>'
   STRIP WHITESPACE))
```

诊断 CREATE INDEX 语句对填充的表发出的 SQL20306N 消息

要确定出现 SQL20306N 错误消息的原因，请参阅中“SQL20305N 和 SQL20306N 错误消息的问题确定”，然后遵循下列步骤:

1. 要在存储的文档中查找失败的节点值，请在 db2diag 日志文件中搜索字符串 SQL20306N 并与原因码编号匹配。 依据原因码，您会找到一组指示信息和生成的 XQuery 语句，可使用它们来找出文档中导致该错误的值。
   - 对于较小的节点值，请在 XQuery 谓词中使用完整值。
   - 对于节点值太长而无法输出至 db2diag 日志文件的情况，将该值的起始字节与 XQuery 谓词中的 fn:starts-with 函数配合使用，并将该值的结束字节与 fn:ends-with 函数配合使用。
2. 执行 XQuery 语句以检索完整文档和包含导致故障的值的文档片段。 为了提供文档中出现错误的位置的上下文信息，XQuery 语句将输出文档片段，并以导致故障的节点值的父代开头。
3. 使用索引 XML 模式来标识要检查的匹配 XML 节点。因为生成的 XQuery 语句对名称空间使用通配符，所以要限定的多个问题值可能具有不同名称空间（但不常见）。如果发生这种情况，那么必须在索引 XML 模式中使用名称空间声明来确定匹

配 XML 节点的正确集合。如果未在谓词中使用完整值来过滤结果，那么必须使用索引 XML 模式来验证 XQuery 语句返回的限定问题值。

4. 一旦在文档中找到失败的值，则应修改 CREATE INDEX XML 模式来更正该问题，或使用 XQuery 谓词来更新或删除包含失败值的文档。

示例: CREATE INDEX 失败

在此示例中，已存储文档中的限定文本值超出索引 XML 模式中的 VARCHAR(4) 长度约束，所以 CREATE INDEX 语句失败。对于较大的值，生成的 XQuery 会在谓词中使用 fn:starts-with 和 fn:ends-with 函数。注意，*N 用作错误消息中模式信息不适用的位置的占位符。

```
INSERT INTO t VALUES (XMLPARSE (DOCUMENT '
   <x>This is the beginning of the document
       <y>test
          <z>rld123456789012345678901234123456789012345678901234
       456789012345678901234567890098765432112345678900987665
       4321123445677880987654321123456789O455</z>
       </y>
   </x>' strip whitespace))
```

```
DB20000I 已成功完成 SQL 命令。
```

```
CREATE INDEX i1 ON t(x)
   GENERATE KEY USING XMLPATTERN '/x/y//text()'
   AS SQL VARCHAR(4)
```

```
DB21034E  该命令被当作 SQL 语句来处理，因为它是无效的"命令行处理器"命令。在 SQL 处理期间，它返回:

SQL20306N  由于在将 XML 值插入到索引中时检测到错误，所以无法创建一个 XML 列的
索引。原因码为"1"。
对于 XML 模式的相关原因码，XML 标识为"*N"，XML
模式数据类型为"*N"。SQLSTATE=23526
```

db2diag 日志文件中的输出如下所示（包含少许格式更改）：

```
2007-03-06-12.08.48.437571-480 I10148A1082        级别: 警告
PID     : 1544348              TID  : 1801        PROC : db2sysc
INSTANCE: adua                 NODE : 000         DB   : ADTEST
APPHDL  : 0-30                 APPID: *LOCAL.adua.070306200844
AUTHID  : ADUA
EDUID   : 1801                 EDUNAME: db2agent (ADTEST)
函数: DB2 UDB, XML 存储器和索引管理器,
          xmlsIkaProcessErrorMsg, 探测: 361
消息: ZRC=0x80A50412=-2136669166=XMS_XML_CRIX_ERROR
          "创建 XML 索引时发生 XML 节点值错误"
DATA #1: 字符串, 36 字节
SQL 代码: SQL20306N; 原因码: 1
DATA #2: 字符串, 72 字节
要在文档中查找导致错误的值，请执行以下 XQuery。
DATA #3: 字符串, 435 字节
xquery for $doc in db2-fn:xmlcolumn("ADUA.T.X")[/*:x/*:y/*:z/text()
[fn:starts-with(., "rld123456789012345678901234123456789012345678901234567890123")
and fn:ends-with(., "5678900987654321123445677880987654321123456789O455")]]
return
<Result>
 <ProblemDocument> {$doc} </ProblemDocument>
 <ProblemValue> {$doc/*:x/*:y/*:z/text()/..} </ProblemValue>
</Result>;
```

查询语句的结果如下所示（包含少许格式更改）：

```
<Result>
    <ProblemDocument>
         <x>This is the beginning of the document
           <y>test
              <z>rld123456789012345678901234123456789012345678901234567890123
       4567890123456789012345678900987654321123456789009876543211234456778809876543211234567890455</z>
          </y>
       </x>
</ProblemDocument>
 <ProblemValue>
     <z>rld123456789012345678901234123456789012345678901234567890123
     4567890123456789012345678900987654321123456789009876543211234456778809876543211234567890455</z>
 </ProblemValue>
</Result>
```

更正以下错误:

可更改 CREATE INDEX XML 模式以提高最大 VARCHAR 长度:

```
CREATE INDEX i1 ON t(x)
   GENERATE KEY USING XMLPATTERN '/x/y//text()'
   AS SQL VARCHAR(200)
```

第 7 章 更新 XML 数据

要更新 XML 列中的数据，可使用 SQL UPDATE 语句。想要更新特定行时，可包括 WHERE 子句。将替换整个列值。XML 列的输入必须是格式良好的 XML 文档。应用程序数据类型可以是 XML、字符或二进制类型。

更新 XML 列时，还可能想要针对已注册的 XML 模式验证输入 XML 文档。可以使用 XMLVALIDATE 函数来执行此操作。

可以使用 XML 列值来指定要更新的行。要查找 XML 文档内的值，需要使用 XQuery 表达式。可使用 XMLEXISTS 谓词指定 XQuery 表达式，它允许您指定 XQuery 表达式并确定该表达式是否导致空序列。在 WHERE 子句中指定了 XMLEXISTS 子句时，如果 XQuery 表达式返回非空序列，那么将更新行。

下列示例说明了如何更新 XML 列中的 XML 数据。这些示例使用表 MYCUSTOMER，它是样本 CUSTOMER 表的副本。这些样本假定 MYCUSTOMER 已包含客户标识值为 1004 的行。假定用于更新现有列数据的 XML 数据存储在文件 c7.xml 中，其内容如下所示:

```
<customerinfo Cid="1004">
  <name>Christine Haas</name>
  <addr country="Canada">
    <street>12 Topgrove</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9Y-8G9</pcode-zip>
  </addr>
  <phone type="work">905-555-5238</phone>
  <phone type="home">416-555-2934</phone>
</customerinfo>
```

示例: 在 JDBC 应用程序中，以二进制数据的形式读取文件 c7.xml 中的 XML 数据，并使用它来更新 XML 列中的数据:

```
PreparedStatement updateStmt = null;
String sqls = null;
int cid = 1004;
sqls = "UPDATE MyCustomer SET Info=? WHERE Cid=?";
updateStmt = conn.prepareStatement(sqls);
updateStmt.setInt(1, cid);
File file = new File("c7.xml");
updateStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
updateStmt.executeUpdate();
```

示例: 在嵌入式 C 应用程序中，通过二进制 XML 主变量更新 XML 列中的数据:

```
  EXEC SQL BEGIN DECLARE SECTION;
      sqlint64 cid;
      SQL TYPE IS XML AS BLOB (10K) xml_hostvar;
  EXEC SQL END DECLARE SECTION;
...
cid=1004;
/* Read data from file c7.xml into xml_hostvar */
...
EXEC SQL UPDATE MyCustomer SET Info=:xml_hostvar WHERE Cid=:cid;
```

在这些示例中，<customerinfo> 元素内 Cid 属性的值正好也存储在 CID 关系列中。因此，UPDATE 语句中的 WHERE 子句使用关系列 CID 来指定要更新的行。如果仅在 XML 文档内找到用于确定选择哪些行来进行更新的值，那么可以使用 XMLEXISTS 谓词。例如，可以按如下所示将上一个嵌入式 C 应用程序示例中的 UPDATE 语句更改为使用 XMLEXISTS:

```
EXEC SQL UPDATE MyCustomer SET Info=:xml_hostvar
       WHERE XMLEXISTS ('$doc/customerinfo[@Cid = $c]'
                        passing INFO as "doc", cast(:cid as integer) as "c");
```

示例: 以下示例更新 MYCUSTOMER 表中的现有 XML 数据。SQL UPDATE 语句会作用于 MYCUSTOMER 表的一行并将该行 INFO 列中的文档替换为变换表达式修改后的文档逻辑快照:

```
UPDATE MyCustomer
SET info = XMLQUERY(
   'transform
    copy $newinfo := $info
    modify do insert <status>Current</status> as last into $newinfo/customerinfo
return $newinfo' passing info as "info")
WHERE cid = 1004
```

在变换表达式中使用更新操作

必须在变换表达式的 **modify** 子句中使用 DB2 XQuery 更新表达式。更新表达式会作用于变换表达式 **copy** 子句创建的复制节点。

下列表达式是更新表达式:

- 删除表达式
- 插入表达式
- 重命名表达式
- 替换表达式
- 其 **return** 子句包含更新表达式的 FLWOR 表达式
- 其 **then** 或 **else** 子句包含更新表达式的条件表达式
- 用逗号隔开的两个或多个更新表达式，其中所有操作数都是更新表达式或空序列

DB2 XQuery 会对无效更新表达式返回错误。例如，如果条件表达式的一个分支包含更新表达式，而另一个分支包含并非空序列的非更新表达式，那么 DB2 XQuery 会返回错误。

变换表达式并非更新表达式，原因是它不会修改任何现有节点。变换表达式会创建现有节点的修改副本。变换表达式的结果可能包括通过更新变换表达式 **modify** 子句中的表达式创建的节点以及先前存在的节点。

处理 XQuery 更新操作

在变换表达式中，**modify** 子句可指定多个更新。例如，**modify** 子句可包含两个更新表达式，一个表达式用于替换现有值，另一个表达式用于插入新元素。**modify** 子句包含多个更新表达式时，每个更新表达式会独立求值，并生成由变换表达式 **copy** 子句创建的特定节点的关联更改操作列表。

在 **modify** 子句中，更新表达式不能修改由其他更新表达式添加的新节点。例如，如果一个更新表达式添加了新的元素节点，那么另一个更新表达式不能更改新创建节点的名称。

变换表达式 **modify** 子句中指定的所有更改操作会被收集到一起，并按以下顺序进行有效应用:

1. 下列更新操作是按不确定的顺序执行的:
   - 未使用 **before**、**after**、**as first** 或 **as last** 之类的排序关键字的插入操作。
   - 所有重命名操作。
   - 替换操作，其中指定了关键字 **value of** 并且目标节点为属性、文本、注释或处理指令节点。
2. 使用 **before**、**after**、**as first** 或 **as last** 之类的排序关键字的插入操作。
3. 未指定关键字 **value of** 的替换操作。
4. 替换操作，其中指定了关键字 **value of** 并且目标节点为元素节点。
5. 所有删除操作。

应用更改操作的顺序应确保一系列多个更改将生成确定的结果。有关更新操作的执行顺序如何保证一系列多个更改生成确定结果的示例，请参阅『示例』中的最后一个 XQuery 表达式。

无效 XQuery 更新操作

处理变换表达式期间，如果出现下列其中一种情况，那么 DB2 XQuery 会返回错误:

- 对同一节点应用了两个或多个重命名操作。
- 对同一节点应用了使用关键字 **value of** 的两个或多个替换操作。
- 对同一节点应用了未使用关键字 **value of** 的两个或多个替换操作。
- 变换表达式的结果并非有效 XDM 实例。

  包含的元素的两个属性同名就是一个无效 XDM 实例的示例。
- XDM 实例包含不一致的名称空间绑定。

  下面是不一致的名称空间绑定的示例:
  - 属性节点的 QName 中的名称空间绑定与其父元素节点中的名称空间绑定不一致。
  - 具有同一父代的两个属性节点中的名称空间绑定相互不一致。

示例

在以下示例中，变换表达式的 **copy** 子句将变量 $product 绑定至元素节点的副本，而变换表达式的 **modify** 子句使用两个更新表达式来更改复制节点:

```
xquery
transform
copy $product := db2-fn:sqlquery(
  "select description from product where pid='100-100-01'")/product
modify(
  do replace value of $product/description/price with 349.95,
  do insert <status>Available</status> as last into $product )
return $product
```

以下示例在 SQL UPDATE 语句中使用 XQuery 变换表达式来修改 CUSTOMER 表中的 XML 数据。SQL UPDATE 语句会作用于 CUSTOMER 表的某行。变换表达式会根据该行的 INFO 列创建 XML 文档的副本，并将 `status` 元素添加至文档副本。UPDATE 语句会将该行 INFO 列中的文档替换为变换表达式修改后的文档副本:

```
UPDATE customer
SET info = xmlquery( 'transform
   copy $newinfo := $info
   modify do insert <status>Current</status> as last into $newinfo/customerinfo
   return $newinfo' passing info as "info")
WHERE cid = 1003
```

以下示例使用 DB2 SAMPLE 数据库中的 CUSTOMER 表。在 CUSTOMER 表中，XML 列 INFO 包含客户地址和电话信息。

在以下示例中，SQL SELECT 语句会作用于 CUSTOMER 表的某行。变换表达式的 **copy** 子句会根据列 INFO 创建 XML 文档的副本。删除表达式会删除文档副本中的地址信息和非工作电话号码。**return** 会使用 CUSTOMER 表的原始文档中的客户标识属性和国家或地区属性:

```
SELECT XMLQUERY( 'transform
  copy $mycust := $d
  modify
    do delete ( $mycust/customerinfo/addr,
       $mycust/customerinfo/phone[@type != "work"] )
  return
  <custinfo>
    <Cid>{data($d/customerinfo/@Cid)}</Cid>
    {$mycust/customerinfo/*}
    <country>{data($d/customerinfo/addr/@country)}</country>
  </custinfo>'
  passing INFO as "d")
FROM CUSTOMER
WHERE CID = 1003
```

对 SAMPLE 数据库运行时，该语句将返回以下结果:

```
<custinfo>
  <Cid>1003</Cid>
    <name>Robert Shoemaker</name>
    <phone type="work">905-555-7258</phone>
  <country>Canada</country>
</custinfo>
```

在以下示例中，XQuery 表达式会说明更新操作的顺序如何保证一系列多个更改将生成确定的结果。插入表达式会在 `phone` 元素后添加 `status` 元素，而替换表达式会将 `phone` 元素替换为 `email` 元素:

```
xquery
let $email  := <email>jnoodle@my-email.com</email>
let $status := <status>current</status>
return
  transform
  copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1002')
  modify (
    do replace $mycust/customerinfo/phone with $email,
    do insert $status after $mycust/customerinfo/phone[@type = "work"] )
  return $mycust
```

在 **modify** 子句中，替换表达式在插入表达式之前。但是，更新复制节点序列 `$mycust` 时，会在替换更新操作之前执行插入更新操作，以确保生成确定的结果。对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<customerinfo Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <email>jnoodle@my-email.com</email>
  <status>current</status>
</customerinfo>
```

如果先执行替换操作，那么 phone 元素不会在节点序列中，而用于在 phone 元素后插入 status 元素的操作没有任何意义。

有关更新操作的顺序的信息，请参阅第 166 页的『处理 XQuery 更新操作』。

使用其他表中的信息更新 XML 文档

可使用其他数据库列中的数据来更新 XML 文档。例如，如果某个表包含更新后的客户信息，那么可使用 SQL/XML 语句和 XQuery 表达式来更新 XML 文档中的客户信息。

使用以下 SQL 语句来创建包含新的客户电话号码的样本表。

```
CREATE TABLE NewPhones (
   CID BIGINT NOT NULL PRIMARY KEY, PhoneNo VARCHAR(20), Type VARCHAR(10))~

INSERT INTO NewPhones (CID, PhoneNo, Type) VALUES (1001, '111-222-3333', 'cell' )~
INSERT INTO NewPhones (CID, PhoneNo, Type) VALUES (1002, '222-555-1111', 'home' )~
INSERT INTO NewPhones (CID, PhoneNo, Type) VALUES (1003, '333-444-2222', 'home' )~
```

使用以下 SQL 语句来更新 CUSTOMER 表中的客户电话号码。该语句使用 XMLQUERY 函数和 XQuery 表达式，该表达式由 FLWOR 表达式和包含插入表达式的变换表达式组成。

```
UPDATE CUSTOMER SET INFO = XMLQUERY(
   'let $myphone := db2-fn:sqlquery(''SELECT XMLELEMENT(Name "phone",
        XMLATTRIBUTES( NewPhones.Type as "type" ), NewPhones.PhoneNo )
        FROM NewPhones WHERE CID = parameter(1)'', $mycid )
   return
     transform
     copy $mycust := $d
     modify
      do insert $myphone after $mycust/customerinfo/phone[last()]
     return
       $mycust'
  passing INFO as "d", 1002 as "mycid" )
WHERE CID = 1002~
```

XMLQUERY 函数会执行 XQuery 表达式，该表达式会将 phone 元素节点添加至客户信息并将修改后的信息返回至 UPDATE 语句。在 XQuery FLWOR 表达式的 **let** 子句中，db2-fn:sqlquery 函数会执行 SQL fullselect 语句。全查询会使用从 XMLQUERY 传递至 XQuery 表达式的客户标识。

FULLSELECT 语句必须返回 XML 数据类型。为了根据 SELECT 语句返回的 PHONENO 和 TYPE 数据创建 XML 数据类型，XMLELEMENT 和 XMLATTRIBUTES 函数根据提供的数据创建了 phone 元素节点。

在此示例中，**let** 子句中的 db2-fn:sqlquery 执行的全查询会创建以下 phone 元素节点。

```
<phone type="home">222-555-1111<phone>
```

运行以下 SQL SELECT 语句以查看客户信息，现在同时包含工作电话号码和家庭电话号码。

```
SELECT INFO FROM CUSTOMER WHERE CID = 1002~
```

从表中删除 XML 数据

要删除包含 XML 文档的行，可使用 DELETE SQL 语句。如果要删除特定行，那么包括 WHERE 子句。

可以根据 XML 列中的值指定要删除的行。要查找 XML 文档内的值，需要使用 XQuery 表达式。可使用 XMLEXISTS 谓词指定 XQuery 表达式，它允许您指定 XQuery 表达式并确定该表达式是否导致空序列。在 WHERE 子句中指定了 XMLEXISTS 的情况下，如果 XQuery 表达式返回非空序列，那么将删除行。

XML 列必须为空或包含格式良好的 XML 文档。如果要从 XML 列中删除 XML 文档但不删除行，那么使用带有 SET NULL 的 UPDATE SQL 语句，以将列设置为空（如果列定义为可空）。要删除现有 XML 文档中属性或元素之类的对象，请使用 UPDATE SQL 语句及 XQuery 更新表达式。XQuery 更新表达式可更改现有 XML 文档的副本。然后，UPDATE 语句会将 XQuery 更新表达式返回的已更改副本应用于指定行的 XML 列。

下列示例说明了如何从 XML 列中删除 XML 数据。这些示例使用表 MyCustomer（它是样本 Customer 表的副本），并假定 MyCustomer 中已填充所有 Customer 数据。

示例: 删除表 MyCustomer 中 Cid 列值为 1002 的行。

```
DELETE FROM MyCustomer WHERE Cid=1002
```

示例: 删除表 MyCustomer 中 `city` 元素的值为 `Markham` 的行。此语句删除客户标识为 1002 的行。

```
DELETE FROM MyCustomer
 WHERE XMLEXISTS ('$d//addr[city="Markham"]' passing INFO as "d")
```

示例: 删除 MyCustomer 中 `city` 元素的值为 `Markham` 的行中的 XML 文档，但保留该行。此语句应删除客户标识为 1002 的行的 Info 列中的 XML 数据。

```
UPDATE MyCustomer SET Info = NULL
  WHERE XMLEXISTS ('$d//addr[city="Markham"]' passing INFO as "d")
```

示例: 以下示例删除 MyCustomer 表中现有 XML 数据内的电话信息。SQL UPDATE 语句会作用于 MyCustomer 表的某行。XQuery 变换表达式通过该行的 INFO 列创建 XML 文档的副本，并使用 XQuery 删除表达式除去文档副本中的工作电话号码。UPDATE 语句会将该行 INFO 列中的文档替换为变换表达式修改后的文档副本:

```
UPDATE MyCustomer
SET info = XMLQUERY(
   'transform
    copy $newinfo := $info
    modify do delete ($newinfo/customerinfo/phone[@type="work"])
    return $newinfo' passing info as "info")
WHERE cid = 1004
```

第 8 章 XML 模式存储库

XML 模式存储库（XSR）是用于处理 XML 列中存储的 XML 实例文档的所有 XML 工件的存储库。XSR 的用途是支持您执行依赖于这些 XML 工件的任务。

XML 实例文档可能引用了指向相关联 XML 模式、DTD 或其他外部实体的统一资源标识（URI）。必须使用此 URI 来处理实例文档。 DB2 数据库系统使用 XSR 管理对这种外部引用的 XML 工件的依赖关系，而不需要更改 URI 位置引用。

如果没有这种机制来存储相关联 XML 模式、DTD 或外部实体，那么外部资源在数据库需要时可能无法访问，或者可能被更改，但未同时触发对存储在数据库内的已验证和带注释的 XML 文档的必需更改。使用 XSR 还可以避免查找外部文档所需的其他开销以及可能对性能产生的影响。

每个数据库都包含一个位于数据库目录中并且由目录表、目录视图和一些系统定义的存储过程组成的 XML 模式存储库，以将数据输入到这些目录表中。

XSR 对象

XML 模式存储库（XSR）支持创建一份信息，作为 XSR 对象包含在 XML 模式、DTD 或外部实体中。此信息用来验证和处理 XML 列中存储的 XML 实例文档。

通过注册过程（用于标识 XML 模式、DTD 或外部实体）使用新的 XSR 对象之前，必须将其显式添加至 XSR。可以从 Java 应用程序、存储过程或命令行处理器来注册 XSR 对象。

使用的最多的 XSR 对象是 XML 模式。XSR 中的每个 XML 模式都可以由一个或多个 XML 模式文档组成。在 XML 模式由多个文档组成的情况下，用于开始注册过程的文档是 XML 模式主文档。在 XML 模式只由一个文档组成的情况下，该文档就是 XML 模式主文档。

有关 XSR 存储过程和命令的语法描述，请参阅第 423 页的附录 C，『XSR 存储过程和命令』。

XSR 对象注册

在将 XML 模式、DTD 或外部实体用于处理 XML 文档之前，它必须先向 XML 模式存储库（XSR）注册。向 XSR 注册时会创建一个 XSR 对象。

要注册大多数 XML 模式，需要增大应用程序堆大小（applheapsz）配置参数。要在 Windows 32 位操作系统上注册非常复杂的 XML 模式，还需要增大代理程序堆栈大小（agent_stack_sz）配置参数。有关如何更改这些配置参数的信息，请参阅下面的相关链接。

对于 XML 模式，XSR 对象注册涉及到下列步骤:

1. 在 XML 模式存储库中注册 XML 模式主文档。

2. 指定要随 XSR 对象一起包括的其他 XML 模式文档。仅当 XML 模式由多个模式文档组成时，此步骤才是必需的。
3. 完成向 XML 模式存储库进行注册的过程。

对于 DTD 和外部实体，向 XML 模式存储库注册 XSR 对象是一个只包含单个步骤的过程。

可以通过下面任何一项来执行 XSR 对象注册步骤:
- Java 应用程序
- 存储过程
- 命令行处理器

注意，因为不能通过 CLP 命令传递必需的文件信息，所以不能使用这些命令从主机应用程序注册 XML 模式。为了从通过 CLI/ODBC 或 JDBC 驱动程序连接至 DB2 数据库的应用程序注册 XML 模式，请使用存储过程方法。

在下面这些方法的描述中，XML 模式的示例由使用的两个 XML 模式文档组成: “PO.xsd”和“address.xsd”，这两个文档都存储在本地 C:\TEMP 中。用户要用包含两部分的 SQL 名称“user1.POschema”注册此模式。XML 模式有一个与它关联的属性文件，称为 schemaProp.xml。此属性文件也存储在同一个本地 C:\TEMP 目录中。这两个 XML 模式文档没有与它们关联的属性。用户定义了 URI，通过它此模式在外部称为“http://myPOschema/PO”。

特权

任何具有 DBADM 权限的用户都可以注册 XSR 对象。对于所有其他用户，特权取决于注册过程中提供的 SQL 模式。如果 SQL 模式不存在，那么需要具有对数据库的 IMPLICIT_SCHEMA 权限才能注册模式。如果 SQL 模式存在，那么注册模式的用户需要具有对 SQL 模式的 CREATEIN 特权。

对于 XML 模式，启动 XSR 对象注册过程（例如，通过 XSR_REGISTER 存储过程来注册）的用户还必须指定其他 XML 模式文档（如果适当）并且完成注册过程。

将自动授予 XSR 对象的创建者对 XSR 对象的 USAGE 特权。

通过存储过程注册 XSR 对象

创建数据库时，还会创建用于注册 XML 模式的存储过程。要通过存储过程方法注册 XML 模式，请使用 CALL 语句来调用 XSR_REGISTER、XSR_ADDSCHEMADOC 和 XSR_COMPLETE 存储过程。

注册或添加文档时，不会检查 XML 模式文档的准确性。只有在完成 XML 模式注册时才执行文档检查。

注册 XML 模式:

1. 通过调用 SYSPROC.XSR_REGISTER 存储过程来注册 XML 模式主文档:

```
CALL SYSPROC.XSR_REGISTER ('user1', 'POschema', 'http://myPOschema/PO',
                                                  :content_host_var, NULL)
```

2. 完成注册之前，添加要随主 XML 模式一起包括的任何其他 XML 模式文档。注意，每个附加模式文档只能包括一次。对于示例来说，此步骤不是可选的，因为 XML 模

式由两个 XML 模式文档组成，这两个文档都必须已经注册。使用 XSR_ADDSCHEMADOC 存储过程来添加其他 XML 模式文档。在以下示例中，将 address 的模式构造添加至 XSR 对象:

```
CALL SYSPROC.XSR_ADDSCHEMADOC ('user1', 'POschema', 'http://myPOschema/address',
                                                :content_host_var, NULL)
```

3. 通过调用 SYSPROC.XSR_COMPLETE 存储过程来完成注册。在以下示例中，最后一个参数指示不会将 XML 模式用于分解（如果值为 1，那么指示它将用于分解）:

```
CALL SYSPROC.XSR_COMPLETE ('user1', 'POschema', :schemaproperty_host_var, 0)
```

特权

具有 DBADM 权限的任何用户都可以注册 XML 模式。对于所有其他用户，特权取决于注册过程中提供的 SQL 模式。如果 SQL 模式不存在，那么需要具有对数据库的 IMPLICIT_SCHEMA 权限才能注册模式。如果 SQL 模式存在，那么注册模式的用户需要具有对 SQL 模式的 CREATEIN 特权。

将自动授予 XSR 对象的创建者对 XSR 对象的 USAGE 特权。

通过命令行处理器注册 XSR 对象

要通过命令行处理器注册 XML 模式，请使用 REGISTER XMLSCHEMA、ADD XMLSCHEMA DOCUMENT 和 COMPLETE XMLSCHEMA 命令。

注册或添加文档时，不会检查 XML 模式文档的准确性。只有在完成模式注册时才执行文档检查。

注意，因为不能通过 CLP 命令传递必需的文件信息，所以不能使用这些命令从主机应用程序注册 XML 模式。为了从通过 CLI/ODBC 或 JDBC 驱动程序连接至 DB2 数据库的应用程序注册 XML 模式，请使用存储过程方法。

注册 XML 模式:

1. 通过发出 REGISTER XMLSCHEMA 命令来注册 XML 模式主文档:

```
REGISTER XMLSCHEMA 'http://myPOschema/PO'
FROM 'file://c:/TEMP/PO.xsd'
AS user1.POschema
```

2. 完成注册之前，可以选择添加要随主 XML 模式一起包括的其他 XML 模式文档。使用 ADD XMLSCHEMA DOCUMENT 命令来添加其他 XML 模式文档。注意，每个附加模式文档只能包括一次。在以下示例中，将 address 的模式构造添加至存储器:

```
ADD XMLSCHEMA DOCUMENT TO user1.POschema
   ADD 'http://myPOschema/address'
   FROM 'file://c:/TEMP/address.xsd'
```

3. 通过发出 COMPLETE XMLSCHEMA 命令来完成注册:

```
COMPLETE XMLSCHEMA user1.POschema
WITH 'file://c:TEMP/schemaProp.xml'
```

特权

具有 DBADM 权限的任何用户都可以注册 XML 模式。对于所有其他用户，特权取决于注册过程中提供的 SQL 模式。如果 SQL 模式不存在，那么需要具有对数据库的 IMPLICIT_SCHEMA 权限才能注册模式。如果 SQL 模式存在，那么注册模式的用户需要具有对 SQL 模式的 CREATEIN 特权。

将自动授予 XSR 对象的创建者对 XSR 对象的 USE 特权。

对 XML 模式注册和删除的 Java 支持

IBM 数据服务器 JDBC 和 SQLJ 驱动程序 提供了一些方法，它们允许您编写 Java 应用程序来注册和除去 XML 模式及其组件。

方法包括:

DB2Connection.registerDB2XMLSchema
: 在 DB2 中注册一种使用一个或多个 XML 模式文档的 XML 模式。此方法有两种格式：一种格式用于从 InputStream 对象输入的 XML 模式文档，另一种格式用于采用 `String` 的 XML 模式文档。

DB2Connection.deregisterDB2XMLObject
: 从 DB2 中除去 XML 模式定义。

DB2Connection.updateDB2XmlSchema
: 将已注册 XML 模式中的 XML 模式文档替换为另一个已注册 XML 模式中的 XML 模式文档。可以选择删除复制了其内容的 XML 模式。此方法只能用于与 DB2 数据库 Linux 版、UNIX 版和 Windows 版的连接。

在调用这些方法之前，必须将支持这些方法的存储过程安装在 DB2 数据库服务器上。

示例: 注册 XML 模式: 以下示例说明如何使用 registerDB2XmlSchema 在 DB2 中注册一种模式，该模式使用从输入流中读取的单个 XML 模式文档（customer.xsd）。已注册的模式的 SQL 模式名称为 SYSXSR。*xmlSchemaLocations* 值为空，因此 DB2 在调用提供了非空 XML 模式位置值的 DSN_XMLVALIDATE 时找不到此 XML 模式。未注册其他属性。

图 6. 向 DB2 注册一种 XML 模式的示例，该模式使用输入流中的 XML 文档

```
public static void registerSchema(
    Connection con,
    String schemaName)
    throws SQLException {
    // Define the registerDB2XmlSchema parameters
  String[] xmlSchemaNameQualifiers = new String[1];
  String[] xmlSchemaNames = new String[1];
  String[] xmlSchemaLocations = new String[1];
  InputStream[] xmlSchemaDocuments = new InputStream[1];
  int[] xmlSchemaDocumentsLengths = new int[1];
  java.io.InputStream[] xmlSchemaDocumentsProperties = new InputStream[1];
  int[] xmlSchemaDocumentsPropertiesLengths = new int[1];
    InputStream xmlSchemaProperties;
    int xmlSchemaPropertiesLength;
    //Set the parameter values
  xmlSchemaLocations[0] = "";
    FileInputStream fi = null;
  xmlSchemaNameQualifiers[0] = "SYSXSR";
```

```
  xmlSchemaNames[0] = schemaName;
      try {
        fi = new FileInputStream("customer.xsd");
    xmlSchemaDocuments[0] = new BufferedInputStream(fi);
    } catch (FileNotFoundException e) {
                  e.printStackTrace();
  }
      try {
     xmlSchemaDocumentsLengths[0] = (int) fi.getChannel().size();
     System.out.println(xmlSchemaDocumentsLengths[0]);
    } catch (IOException e1) {
                  e1.printStackTrace();
  }
  xmlSchemaDocumentsProperties[0] = null;
  xmlSchemaDocumentsPropertiesLengths[0] = 0;
    xmlSchemaProperties = null;
    xmlSchemaPropertiesLength = 0;
    DB2Connection ds = (DB2Connection) con;
    // Invoke registerDB2XmlSchema
    ds.registerDB2XmlSchema(
        xmlSchemaNameQualifiers,
        xmlSchemaNames,
        xmlSchemaLocations,
        xmlSchemaDocuments,
        xmlSchemaDocumentsLengths,
        xmlSchemaDocumentsProperties,
        xmlSchemaDocumentsPropertiesLengths,
        xmlSchemaProperties,
        xmlSchemaPropertiesLength,
        false);
}
```

示例: 除去 XML 模式: 以下示例说明如何使用 deregisterDB2XmlObject 从 DB2 中除去 XML 模式。已注册的模式的 SQL 模式名称为 SYSXSR。

图 7. 有关从 DB2 中除去 XML 模式的示例

```
public static void deregisterSchema(
    Connection con,
    String schemaName)
    throws SQLException {
    // Define and assign values to the deregisterDB2XmlObject parameters
  String xmlSchemaNameQualifier = "SYSXSR";
    String xmlSchemaName = schemaName;
    DB2Connection ds = (DB2Connection) con;
    // Invoke deregisterDB2XmlObject
    ds.deregisterDB2XmlObject(
        xmlSchemaNameQualifier,
        xmlSchemaName);
}
```

示例: 更新 XML 模式: 以下示例仅适用于与 DB2 数据库 Linux 版、UNIX 版和 Windows 版的连接。它说明了如何使用 updateDB2XmlSchema 将一种 XML 模式的内容更新为另一种 XML 模式的内容。已复制的模式将保存在存储库中。已注册模式的 SQL 模式名为 SYSXSR。

图 8. 有关更新 XML 模式的示例

```
public static void updateSchema(
    Connection con,
  String schemaNameTarget,
  String schemaNameSource)
```

```
    throws SQLException {
  // Define and assign values to the updateDB2XmlSchema parameters
  String xmlSchemaNameQualifierTarget = "SYSXSR";
  String xmlSchemaNameQualifierSource = "SYSXSR";
  String xmlSchemaNameTarget = schemaNameTarget;
  String xmlSchemaNameSource = schemaNameSource;
  boolean dropSourceSchema = false;
    DB2Connection ds = (DB2Connection) con;
  // Invoke updateDB2XmlSchema
  ds.updateDB2XmlSchema(
    xmlSchemaNameQualifierTarget,
    xmlSchemaNameTarget,
  xmlSchemaNameQualifierSource,
  xmlSchemaNameSource,
    dropSourceSchema);
}
```

改变注册的 XSR 对象

在 XML 模式存储库中注册后，可以改变 XSR 对象以启用或禁用分解、删除该对象或使其与注释关联。此外，还可以授予或撤销对已注册的 XSR 对象的使用特权。

XML 模式存储库用于管理 XML 文档对 XML 模式、DTD 或其他外部实体的依赖关系。必须首先将每个 XML 模式、DTD 或外部实体注册为 XML 模式存储库中的一个新 XSR 对象。

演进 XML 模式

XML 模式存储库（XSR）中注册的 XML 模式可演进出新的兼容 XML 模式，而不必再次验证已存储的 XML 实例文档。仅更新 XSR 中注册的 XML 模式；包括其 URI 标识的已存储 XML 实例文档保持不变。

新的 XML 模式必须与原始 XML 模式兼容才能演进。如果两个模式不兼容，那么 XSR_UPDATE 存储过程或 UPDATE XMLSCHEMA 命令将返回错误，并且不会演进任何模式。请参阅*演进 XML 模式的兼容性要求*。

要在 XSR 中演进 XML 模式:

1. 调用 XSR_REGISTER 存储过程或运行 REGISTER XMLSCHEMA 命令以在 XSR 中注册新 XML 模式。
2. 最后，调用 XSR_UPDATE 存储过程或运行 UPDATE XMLSCHEMA 命令以在 XSR 中更新新 XML 模式。

成功的模式演进将替换原始 XML 模式。演进后只有更新的 XML 模式可用。

演进 XML 模式的兼容性要求

在 XML 模式存储库（XSR）中演进 XML 模式的过程要求原始 XML 模式和用于更新的新 XML 模式足够相似。如果两个 XML 模式不兼容，那么更新会失败并且会生成错误消息。下面是处理更新进程所需的 10 个兼容性条件，包括不满足描述需求的模式示例。

属性内容

在原始 XML 模式中复杂类型内声明或引用的属性必须同时出现在新 XML 模式中。而且，如果要求的属性未包括在原始 XML 模式中，那么它们也不能出现在新 XML 模式中。

示例 1

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
      <xs:attribute name="b" use="optional" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

示例 2

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
  </xs:complexType>
  </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
      <xs:attribute name="b" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

元素内容

在原始 XML 模式中复杂类型内声明或引用的元素必须出现在新 XML 模式中。如果要求的元素未包括在原始 XML 模式中，那么它们也不能出现在新 XML 模式中；只能添加可选元素。

示例 1

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element name="a" type="xs:string"/>
        <xs:element name="b" minOccurs="0" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

示例 2

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
        <xs:element name="b" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

示例 3

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" substitutionGroup="a"/>
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="a"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" type="xs:string"/>
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="a"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

面冲突 新 XML 模式中简单类型的面值必须与原始 XML 模式中定义的简单类型的值范围兼容。

示例 1

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo" >
       <xs:simpleType>
        <xs:restriction base="xs:decimal" />
        </xs:simpleType>
      </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo">
    <xs:simpleType>
      <xs:restriction base="xs:decimal">
        <xs:totalDigits value="7"/>
      </xs:restriction>    </xs:simpleType>
  </xs:element>
</xs:schema>
```

示例 2

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo">
    <xs:simpleType>
      <xs:restriction base="xs:decimal">
        <xs:totalDigits value="7"/>
        <xs:fractionDigits value="3"/>
        <xs:maxInclusive value="300.00"/>
        <xs:minInclusive value="1.0"/>      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
```

```
<xs:element name="foo">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:totalDigits value="5"/>
      <xs:fractionDigits value="2"/>
      <xs:pattern value="(0|1|2|3|4|5|6|7|8|9|\.)*"/>
      <xs:maxInclusive value="100.00"/>
      <xs:minInclusive value="10.00"/>      </xs:restriction>
  </xs:simpleType>
</xs:element>
</xs:schema>
```

不兼容类型

新 XML 模式中的元素或属性的类型不兼容 条件是已插入 XML 文档导致对新模式的验证失败，或者该模式包括的简单类型注释不同于原始 XML 模式中的注释。

示例

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:integer"/>
</xs:schema>
```

混合内容变为未混合内容

如果复杂类型的内容模型在原始 XML 模式中声明为混合，那么在新 XML 模式中只能声明为混合。

示例

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:complexContent mixed="true">
        <xs:restriction base="xs:anyType">
          <xs:attribute name="a" type="xs:string"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:complexContent mixed="false">
        <xs:restriction base="xs:anyType">
          <xs:attribute name="a" type="xs:string"/>
        </xs:restriction>
```

```
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

可拒绝变为不可拒绝

如果原始 XML 模式的元素声明中的可拒绝属性已启用，那么它在新 XML 模式中也必须为已启用。

示例

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" nillable="true" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" nillable="false" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

已除去元素

原始 XML 模式中声明的全局元素必须同时出现在新 XML 模式中，并且不能作为抽象模式。

示例 1

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" type="xs:string"/>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
</xs:schema>
```

示例 2

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" type="xs:string"/>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" abstract="true" type="xs:string"/>
</xs:schema>
```

已除去类型

如果原始 XML 模式包含派生自另一类型的全局类型，那么全局类型必须也包含在新 XML 模式中。

示例

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root" type="t1"/>
  <xs:complexType name="t1">
    <xs:complexContent>
      <xs:extension base="xs:anyType">
        <xs:attribute name="a" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="t2">
    <xs:complexContent>
      <xs:extension base="t1">
        <xs:attribute name="b" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType></xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root" type="t1"/>
  <xs:complexType name="t1">
    <xs:complexContent>
      <xs:extension base="xs:anyType">
        <xs:attribute name="a" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

简单变为复杂

在原始 XML 模式中包含简单内容的复杂类型不能重新定义为在已更新 XML 模式中包含复杂内容。

示例

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="a" type="xs:string"/>
        </xs:extension>
      </xs:simpleContent>    </xs:complexType>
  </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:complexContent base="xs:anyType">
        <xs:extension base="xs:anyType">
          <xs:attribute name="a" type="xs:string"/>
        </xs:extension>
      </xs:complexContent>    </xs:complexType>
  </xs:element>
</xs:schema>
```

简单内容

原始 XML 模式和新 XML 模式中定义的简单类型必须使用相同的基本类型。

示例

原始 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo" >
       <xs:simpleType>
        <xs:restriction base="xs:decimal" />
        </xs:simpleType>
      </xs:element>
</xs:schema>
```

新 XML 模式:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo" >
       <xs:simpleType>
        <xs:restriction base="xs:string" />
        </xs:simpleType>
      </xs:element>
</xs:schema>
```

方案: 演进 XML 模式

以下方案说明演进在 XML 模式存储库（XSR）中注册的 XML 模式的过程。

Jane 是一家商店的经理，职责是维护数据库，其中的所有商店产品列示在若干 XML 文档中。这些 XML 产品列表符合以下模式:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="prodType">
```

```
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="sku" type="xsd:string" />
    <xsd:element name="price" type="xsd:integer" />
  </xsd:sequence>
  <xsd:attribute name="color" type="xsd:string" />
  <xsd:attribute name="weight" type="xsd:integer" />
</xsd:complexType>

<xsd:element name="products">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="product" type="prodType" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

XML 模式一开始是使用以下命令在 XSR 中注册的:

```
REGISTER XMLSCHEMA 'http://product'
FROM 'file://c:/schemas/prod.xsd'
AS STORE.PROD

COMPLETE XMLSCHEMA STORE.PROD
```

注册 XML 模式后，将对其验证 XML 格式的产品列表并将其插入到商店数据库中。

Jane 决定列表除了包含每个产品的名称、库存标识（SKU）和价格之外，还应包含产品描述。Jane 希望更新原始 XML 模式以容纳添加的产品描述，而不是创建新 XML 模式并因此需要对其重新验证所有现有 XML 文档。需要在原始 XML 模式中添加新的“description”元素:

```
<xsd:complexType name="prodType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="sku" type="xsd:string" />
    <xsd:element name="price" type="xsd:integer" />
    <xsd:element name="description" type="xsd:string" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="color" type="xsd:string" />
  <xsd:attribute name="weight" type="xsd:integer" />
</xsd:complexType>
```

在要插入的 XML 模式段中，“minOccurs”属性设置为“0”。这一点非常重要，否则“description”会成为内容模型中的必需元素，而针对原始模式进行了验证并插入到数据库表中的所有现有 XML 文档不再处于有效状态。要演进 XML 模式，原始模式和新模式必须兼容。有关详细信息，请参阅*演进 XML 模式的兼容性要求*。

必须先在 XSR 中注册新的 XML 模式，才能进行更新:

```
REGISTER XMLSCHEMA 'http://newproduct'
FROM 'file://c:/schemas/newprod.xsd'
AS STORE.NEWPROD

COMPLETE XMLSCHEMA STORE.NEWPROD
```

Jane 现在使用 XSR_UPDATE 存储过程执行更新:

```
CALL SYSPROC.XSR_UPDATE(
  'STORE',
  'PROD',
  'STORE',
  'NEWPROD',
  1)
```

已演进原始 XML 模式。在 XML 实例文档（先前针对 XML 模式 STORE.PROD 验证了这些文档）的 XSR 中管理的所有外部依赖关系根据 XML 模式 STORE.NEWPROD 的内容进行了更新。因为 *dropnewschema* 参数是通过传递非零值设置的，所以会在更新原始模式后删除新模式 STORE.NEWPROD。

已针对原始 XML 文档进行了验证的所有现有 XML 文档未因为更新过程而再次验证。在更新期间，将为确认原始 XML 模式和新 XML 模式是否兼容而执行检查，以确保先前针对原始 XML 模式而进行验证的所有文档对新模式同样有效。在以上示例中，要求在新的“description”元素中将“minOccurs”属性设置为“0”，以使两个 XML 模式相兼容。将针对新的 STORE.PROD 更新版本验证在模式演进之后插入的所有 XML 文档，这些文档现在会包含每个商店产品的“description”元素。

抽取 XML 模式信息的示例

列示已向 XSR 注册的 XML 模式

下列示例说明了可以如何通过 SQL 语句来查询已向 XML 模式存储库完全注册的 XML 模式。必须完成注册之后，才能完全注册 XML 模式。

示例 1: 列示所有已注册的 XML 模式

此示例将返回已向 XSR 注册的所有 XML 模式的 SQL 模式和 SQL 标识。

```
SELECT OBJECTNAME, OBJECTSCHEMA
   FROM SYSCAT.XSROBJECTS
   WHERE OBJECTTYPE='S' AND STATUS='C'
```

示例 2: 返回目标名称空间和模式位置

此示例将返回所有已注册的 XML 模式的目标名称空间和模式位置（*targetNamespace* 和 *schemaLocation*）的统一资源标识（URI）。

```
SELECT TARGETNAMESPACE, SCHEMALOCATION
   FROM SYSCAT.XSROBJECTS
   WHERE OBJECTTYPE='S' AND STATUS='C'
```

示例 3: 返回对象信息文档

此示例将返回所有已注册的模式的对象信息文档（*schemaInfo*）。此 XML 文档是在模式注册期间生成的，用于描述组成已向 XSR 注册的 XML 模式的每个 XML 模式文档。

```
SELECT OBJECTINFO
   FROM SYSCAT.XSROBJECTS
   WHERE OBJECTTYPE='S' AND STATUS='C'
```

检索已向 XSR 注册的 XML 模式的所有组成部分

以下示例说明了可以如何从 XML 模式存储库中检索组成已注册的 XML 模式的所有 XML 模式文档。

示例 1: 将返回已注册的 XML 模式的 XML 模式文档以及目标名称空间和模式位置（*targetNamespace* 和 *schemaLocation*）:

```
SELECT COMPONENT, TARGETNAMESPACE, SCHEMALOCATION
   FROM SYSCAT.XSROBJECTCOMPONENTS
   WHERE OBJECTSCHEMA  = ? AND OBJECTNAME = ?
```

检索 XML 文档的 XML 模式

以下示例说明了可以如何从 XML 模式存储库中检索与 XML 文档相关联的 XML 模式。

示例 1: 检索 XML 文档的 XML 模式的对象标识:

```
SELECT DOC, XMLXSROBJECTID(DOC)
   FROM T
```

示例 2: 检索 XML 文档的 XML 模式的对象标识和 SQL 标识（由两部分组成）:

```
SELECT XMLXSROBJECTID(DOC),
   CAT.OBJECTSCHEMA, CAT.OBJECTNAME
   FROM T, SYSCAT.XSROBJECTS AS CAT
   WHERE XMLXSROBJECTID(DOC) = CAT.OBJECTID
```

第 9 章 XML 数据移动

LOAD、IMPORT 和 EXPORT 实用程序提供了对 XML 数据移动的支持。ADMIN_MOVE_TABLE 存储过程支持在不使包含 XML 列的表处于脱机状态的情况下移动这些表。

导入 XML 数据

可以使用 IMPORT 实用程序将 XML 文档插入到常规关系表中。只能导入格式良好的 XML 文档。

使用 IMPORT 命令的 XML FROM 选项指定要导入的 XML 文档的位置。XMLVALIDATE 选项指定验证已导入的文档的方式。可以选择通过以下方式验证已导入的 XML 数据：针对用 IMPORT 命令指定的模式，针对源 XML 文档内的模式位置提示所标识的模式，或者通过主数据文件中的 XML 数据说明符所标识的模式。还可以使用 XMLPARSE 选项指定导入 XML 文档时处理空格的方式。`xmlchar` 和 `xmlgraphic` 文件类型修饰符允许您指定已导入的 XML 数据的编码特征。

装入 XML 数据

LOAD 实用程序提供了有效的方式将大量 XML 数据插入到表中。此实用程序还允许 IMPORT 实用程序未提供的特定选项，如从用户定义的游标导入的功能。

与 IMPORT 命令一样，可使用 LOAD 命令指定要装入的 XML 数据的位置、XML 数据的验证选项以及空格的处理方式。与 IMPORT 一样，可使用 `xmlchar` 和 `xmlgraphic` 文件类型修饰符来对已装入 XML 数据指定编码特征。

导出 XML 数据

可以从包括一个或多个 XML 数据类型列的表中导出数据。导出的 XML 数据存储在与包含导出的关系数据的主数据文件不同的位置。导出的主数据文件中用 XML 数据说明符（XDS）表示关于每个导出的 XML 文档的信息。XDS 是一个字符串，它指定存储 XML 文档的系统文件的名称、此文件内 XML 文档的准确位置和长度以及用于验证 XML 文档的 XML 模式。

可以使用 EXPORT 命令的 XMLFILE、XML TO 和 XMLSAVESCHEMA 参数来指定关于如何存储导出的 XML 文档的详细信息。`xmlinsepfiles`、`xmlnodeclaration`、`xmlchar` 和 `xmlgraphic` 文件类型修饰符允许您指定关于导出的 XML 数据的存储位置和编码的更多详细信息。

以联机方式移动表

ADMIN_MOVE_TABLE 存储过程将活动表中的数据移至同名的新表对象中，与此同时这些数据保持联机并可供访问。该表可包括具有 XML 数据类型的一列或多列。如果您认为可用性比成本、空间、移动性能和事务开销重要，请使用联机表移动而不是脱机表移动。

可调用该过程一次或多次，对该过程执行的每个操作执行一次调用。使用多个调用允许您有其他选择，例如，取消移动或控制何时使目标表脱机以进行更新。

有关移动 XML 数据的重要注意事项

导入或导出 XML 数据时，有一些限制、先决条件和提示需要注意。在导入或导出 XML 数据之前查看这些注意事项。

导入或导出 XML 数据时记住以下注意事项:

- 导出的 XML 数据始终存储在与包含导出的关系数据的主数据文件不同的位置。
- 缺省情况下，EXPORT 实用程序采用 Unicode 写入 XML 数据。使用 `xmlchar` 文件类型修饰符以通过字符代码页编写 XML 数据，或使用 `xmlgraphic` 文件类型修饰符以通过 UTF-16（图形代码页）编写 XML 数据，而不考虑应用程序代码页。
- XML 数据可存储在非 Unicode 数据库中，在插入之前，要插入到 XML 列中的数据将从数据库代码页转换为 UTF-8。为避免 XML 解析期间可能引入替换字符，要插入的字符数据应仅由数据库代码页中包含的代码点组成。将 `enable_xmlchar` 配置参数设置为 `no` 会阻止在 XML 解析期间插入字符数据类型，从而限制对未进行代码页转换的数据类型（如 BIT DATA、BLOB 或 XML）执行插入。
- 导入或装入 XML 数据时，假定 XML 数据使用 Unicode，除非要导入的 XML 文档包含的声明标记中包括编码属性。可以使用 `xmlchar` 文件类型修饰符来指示要导入的 XML 文档采用字符代码页编码，而 `xmlgraphic` 文件类型修饰符指示要导入的 XML 文档采用 UTF-16 编码。
- IMPORT 和 LOAD 实用程序拒绝包含格式不当的文档的行。
- 如果对 IMPORT 实用程序或 LOAD 实用程序指定了 XMLVALIDATE 选项，那么将针对其匹配模式验证成功的文档插入到表中时，会使用有关用于验证的模式的信息来注释这些文档。如果行中包含针对其匹配模式验证失败的文档，那么将拒绝这些行。
- 如果对 IMPORT 或 LOAD 实用程序指定了 XMLVALIDATE 选项，并且使用多个 XML 模式来验证 XML 文档，那么可能需要增加目录高速缓存大小配置参数 **catalogcache_sz**。如果增加 **catalogcache_sz** 的值不可行或不可能，那么可将单个导入或装入命令分隔为多个命令以使用较少模式文档。
- 导出指定 XQuery 语句的 XML 数据时，可导出查询和 XPath 数据模型（XDM）实例，这些实例是格式不当的 XML 文档。不能将格式不当的已导出 XML 文档直接导入到 XML 列中，原因是使用 XML 数据类型定义的列只能包含完整的格式良好的 XML 文档。
- 如果要收集统计信息，那么 **CPU_PARALLELISM** 设置会在装入期间降至 1。
- XML 装入操作需要使用共享排序内存才能继续。启用 **SHEAPTHRES_SHR** 或 **INTRA_PARALLEL**，或打开连接集中器。缺省情况下会设置 **SHEAPTHRES_SHR**，所以会在缺省配置中提供共享排序内存。
- 装入包含 XML 列的表时，不能对 LOAD 命令指定 **SOURCEUSEREXIT** 选项或 **SAVECOUNT** 参数。
- 与 LOB 文件一样，使用 LOAD 命令时，XML 文件必须位于服务器端。
- 将 XML 数据装入到分区数据库环境中的多个数据库分区中时，所有数据库分区必须可访问包含 XML 数据的文件。例如，可复制这些文件或创建 NFS 安装以使这些文件可访问。

查询和 XPath 数据模型

可通过使用以 SQL 提供的 XQuery 函数或者通过直接调用 XQuery 来访问数据库表中的 XML 数据。查询和 XPath 数据模型（XDM）的实例可能是格式良好的 XML 文档、节点序列、原子值序列或节点与原子值的任意组合。

可通过 EXPORT 命令将各个 XDM 实例写入一个或多个 XML 文件。

导入和导出时的 LOB 和 XML 文件行为

LOB 和 XML 文件共用导入和导出数据时可使用的一些行为和功能。

导出 导出数据时，如果使用 LOBS TO 选项指定了一个或多个 LOB 路径，那么 EXPORT 实用程序将循环使用这些 LOB 路径，以便将每个连续的 LOB 值写入相应的 LOB 文件。同样，如果使用 XML TO 选项指定了一个或多个 XML 路径，那么 EXPORT 实用程序将循环使用这些 XML 路径，以便将每个连续的 XQuery 和 XPath 数据模型（XDM）实例写入相应的 XML 文件。缺省情况下，LOB 值和 XDM 实例与导出的关系数据将写入同一路径。除非设置了 LOBSINSEPFILES 或 XMLINSEPFILES 文件类型修饰符，否则 LOB 文件和 XML 文件都可以有多个值并置至同一文件。

LOBFILE 选项提供了一种方法来指定 EXPORT 实用程序生成的 LOB 文件的基本名称。同样，XMLFILE 选项也提供了一种方法来指定 EXPORT 实用程序生成的 XML 文件的基本名称。缺省 LOB 文件基本名称是导出的数据文件名称，其扩展名为 `.lob`。缺省 XML 文件基本名称是导出的数据文件名称，其扩展名为 `.xml`。因此，导出的 LOB 文件或 XML 文件的全名由基本名称、接着是填满为三位数的编号扩展名以及 `.lob` 或 `.xml` 扩展名组成。

导入 导入数据时，LOB 位置说明符（LLS）与 XML 目标列兼容，而 XML 数据说明符（XDS）与 LOB 目标列兼容。如果未指定 LOBS FROM 选项，那么假定要导入的 LOB 文件与输入关系数据文件位于同一路径中。同样，如果未指定 XML FROM 选项，那么假定要导入的 XML 文件与输入关系数据文件位于同一路径中。

导出示例

在以下示例中，所有 LOB 值将写入文件 `/mypath/t1export.del.001.lob`，而所有 XDM 实例将写入文件 `/mypath/t1export.del.001.xml`：

```
EXPORT TO /mypath/t1export.del OF DEL MODIFIED BY LOBSINFILE
SELECT * FROM USER.T1
```

在以下示例中，第一个 LOB 值将写入文件 `/lob1/t1export.del.001.lob`，第二个 LOB 值将写入文件 `/lob2/t1export.del.002.lob`，第三个 LOB 值将附加至 `/lob1/t1export.del.001.lob`，第四个 LOB 值将附加至 `/lob2/t1export.del.002.lob`，以此类推：

```
EXPORT TO /mypath/t1export.del OF DEL LOBS TO /lob1,/lob2
MODIFIED BY LOBSINFILE SELECT * FROM USER.T1
```

在以下示例中，第一个 XDM 实例将写入文件 `/xml1/xmlbase.001.xml`，第二个 XDM 实例将写入文件 `/xml2/xmlbase.002.xml`，第三个 XDM 实例将写入 `/xml1/xmlbase.003.xml`，第四个 XDM 实例将写入 `/xml2/xmlbase.004.xml`，以此类推：

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /xml1,/xml2 XMLFILE xmlbase
   MODIFIED BY XMLINSEPFILES SELECT * FROM USER.T1
```

导入示例

对于包含单个 XML 列的“mytable”表和以下 IMPORT 命令:

```
IMPORT FROM myfile.del of del LOBS FROM /lobpath XML FROM /xmlpath
MODIFIED BY LOBSINFILE XMLCHAR replace into mytable
```

如果“myfile.del”包含以下数据:

```
mylobfile.001.lob.123.456/
```

IMPORT 实用程序将尝试从文件 /lobpath/mylobfile.001.lob 中文件偏移量为 123 处开始导入 XML 文档（其长度将为 456 字节）。

由于值由 LOB 位置说明符（LLS）而不是 XML 数据说明符（XDS）引用，因此假定“mylobfile.001.lob”文件位于 LOB 路径而不是 XML 路径中。

由于指定了 XMLCHAR 文件类型修饰符，因此假定文档采用字符代码页编码。

XML 数据说明符

使用 EXPORT、IMPORT 和 LOAD 实用程序移动的 XML 数据必须存储在与主数据文件分开的文件中。主数据文件中用 XML 数据说明符（XDS）表示 XML 数据。

XDS 是表示为 XML 标记（其名称是“XDS”）的字符串，它具有用于描述关于列中实际 XML 数据的信息的属性；这种信息涉及包含实际 XML 数据的文件名，以及该文件内 XML 数据的偏移量和长度。下面描述了 XDS 的属性。

FIL 包含 XML 数据的文件的名称。不能指定命名管道。不支持从命名管道导入或装入 XML 文档。

OFF FIL 属性所指定的文件中 XML 数据的字节偏移量（其中偏移量从 0 开始）。

LEN FIL 属性所指定的文件中 XML 数据的长度（以字节计）。

SCH 用于验证此 XML 文档的 XML 模式的标准 SQL 标识。SQL 标识的模式和名称部分分别作为“OBJECTSCHEMA”和“OBJECTNAME”值存储在与此 XML 模式对应的 SYSCAT.XSROBJECTS 目录表的行中。

XDS 在数据文件中解释为字符字段，并且遵循文件格式的字符列解析行为。例如，对于定界 ASCII 文件格式（DEL），如果字符定界符出现在 XDS 中，那么该字符定界符必须加倍。属性值内的特殊字符（<、>、&、’ 和 ″）必须始终转义。区分大小写的对象名必须放在 " 字符实体之间。

示例

考虑值为 abc&"def".del 的 FIL 属性。要将此 XDS 包括在定界 ASCII 文件（其中字符定界符为 ″ 字符），必须使用两个 ″ 并且特殊字符必须转义。

```
<XDS FIL=""abc&amp;&quot;def&quot;.del"" />
```

以下示例显示 XDS 出现在定界 ASCII 数据文件中时的样式。XML 数据存储在 xmldocs.xml.001 文件中字节偏移量从 100 开始的位置，其长度为 300 字节。因为此 XDS 位于用双引号定界的 ASCII 文件中，所以 XDS 标记本身包含的双引号必须加倍。

```
"<XDS FIL = ""xmldocs.xml.001"" OFF=""100"" LEN=""300"" />"
```

以下示例显示标准 SQL 标识 ANTHONY.purchaseOrderTest。在 XDS 中，区分大小写的标识部分必须放在 " 字符实体间:

```
"<XDS FIL='/home/db2inst1/xmlload/a.xml' OFF='0' LEN='6758'
   SCH='ANTHONY.&quot;purchaseOrderTest&quot;' />"
```

导出 XML 数据

导出 XML 数据时，生成的 QDM（XQuery 数据模型）实例将写入与包含导出的关系数据的主数据文件不同的文件。即使未指定 XMLFILE 和 XML TO 选项亦如此。缺省情况下，导出的 QDM 实例将全部放入同一个 XML 文件中。可以使用 XMLINSEPFILES 文件类型修饰符来指定将每个 QDM 实例写入不同文件。

然而，主数据文件中用 XML 数据说明符（XDS）表示 XML 数据。XDS 是表示为 XML 标记（其名称是“XDS”）的字符串，它具有用于描述关于列中实际 XML 数据的信息的属性；这种信息涉及包含实际 XML 数据的文件名，以及该文件内 XML 数据的偏移量和长度。

可以使用 XML TO 和 XMLFILE 选项指定导出的 XML 文件的目标路径和基本名称。如果指定了 XML TO 或 XMLFILE 选项，那么已导出 XML 文件名的格式（存储在 XDS 的 FIL 属性中）为 xmlfilespec.xxx.xml，其中 xmlfilespec 是对 XMLFILE 选项指定的值，而 xxx 是 EXPORT 实用程序生成的 XML 文件的序号。否则，已导出 XML 文件名的格式为 exportfilename.xxx.xml，其中 exportfilename 是对 EXPORT 命令指定的已导出输出文件的名称，而 xxx 是 EXPORT 实用程序生成的 XML 文件的序号。

缺省情况下，导出的 XML 文件将写入已导出数据文件的路径中。导出的 XML 文件的缺省基本名称包括已导出数据文件的名称、追加的 3 位序号和 .xml 扩展名。

示例

在下列示例中，假定 USER.T1 表包含四列两行:

```
C1 INTEGER
C2 XML
C3 VARCHAR(10)
C4 XML
```

表 31. USER.T1

C1	C2	C3	C4
2	<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to><from> Me</from><heading>note1</heading> <body>Hello World!</body></note>	'char1'	<?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him</to><from> Her</from><heading>note2</heading>< body>Hello World!</body></note>
4	NULL	'char2'	?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">to>Us</to><from> Them</from><heading>note3</heading> <body>Hello World!</body></note>

示例 1

以下命令将定界 ASCII（DEL）格式的 USER.T1 的内容导出到“/mypath/t1export.del”文件中。因为没有指定 XML TO 和 XMLFILE 选项，所以将 C2 和 C4 列中包含的 XML 文档与导出的主文件“/mypath”写入同一路径中。这些文件的基本名称为“t1export.del.xml”。XMLSAVESCHEMA 选项指示将在导出过程中保存 XML 模式信息。

```
EXPORT TO /mypath/t1export.del OF DEL XMLSAVESCHEMA SELECT * FROM USER.T1
```

导出的文件“/mypath/t1export.del”包含:

```
2,"<XDS FIL='t1export.del.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='t1export.del.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='t1export.del.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

导出的 XML 文件“/mypath/t1export.del.001.xml”包含:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him
   </to><from>Her</from><heading>note2</heading><body>Hello World!
   </body></note><?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">
   <to>Us</to><from>Them</from>heading>note3</heading><body>
   Hello World!</body></note>
```

示例 2

以下命令将 DEL 格式的 USER.T1 的内容导出到“t1export.del”文件中。将 C2 和 C4 列中包含的 XML 文档写入“/home/user/xmlpath”路径中。使用基本名称“xmldocs”命名 XML 文件，并将导出的多个 XML 文档写入同一个 XML 文件。XMLSAVESCHEMA 选项指示将在导出过程中保存 XML 模式信息。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs XMLSAVESCHEMA SELECT * FROM USER.T1
```

导出的 DEL 文件“/home/user/t1export.del”包含:

```
2,"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='xmldocs.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='xmldocs.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

导出的 XML 文件“/home/user/xmlpath/xmldocs.001.xml”包含:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00">
   <to>Him</to><from>Her</from><heading>note2</heading><body>
   Hello World!</body></note><?xml version="1.0" encoding="UTF-8" ?>
   <note time="14:00:00"><to>Us</to><from>Them</from><heading>
   note3</heading><body>Hello World!</body></note>
```

示例 3

除了将导出的每个 XML 文档写入不同 XML 文件外，以下命令与示例 2 类似。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLINSEPFILES XMLSAVESCHEMA
SELECT * FROM USER.T1
```

导出的文件“/mypath/t1export.del”包含:

```
2,"<XDS FIL='xmldocs.001.xml' />","char1","XDS FIL='xmldocs.002.xml' />"
4,,"char2","<XDS FIL='xmldocs.004.xml' SCH='S1.SCHEMA_A' />"
```

导出的 XML 文件“/home/user/xmlpath/xmldocs.001.xml”包含:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note>
```

导出的 XML 文件“/home/user/xmlpath/xmldocs.002.xml”包含:

```
?xml version="1.0" encoding="UTF-8" ?>note time="13:00:00">to>Him/to>
   from>Her/from>heading>note2/heading>body>Hello World!/body>
   /note>
```

导出的 XML 文件“/home/user/xmlpath/xmldocs.004.xml”包含:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00"><to>Us</to>
   <from>Them</from><heading>note3</heading><body>Hello World!</body>
   </note>
```

示例 4

以下命令将 XQuery 的结果写入 XML 文件。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLNODECLARATION select
xmlquery( '$m/note/from/text()' passing by ref c4 as "m" returning sequence)
   from USER.T1
```

导出的 DEL 文件“/mypath/t1export.del”包含:

```
"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='3' />"
"<XDS FIL='xmldocs.001.xml' OFF='3' LEN='4' />"
```

导出的 XML 文件“/home/user/xmlpath/xmldocs.001.xml”包含:

```
HerThem
```

注: 此特定 XQuery 的结果不生成结构良好的 XML 文档。因此，不能将上面导出的文件直接导入到 XML 列中。

导入 XML 数据

通过对 DB2 数据库 Linux 版、UNIX 版和 Windows 版数据对象使用表名或昵称，可使用 IMPORT 实用程序将 XML 数据导入到 XML 表列中。

将数据导入到 XML 表列中时，可以使用 XML FROM 选项来指定一个或多个输入 XML 数据文件的路径。例如，对于先前已导出的 XML 文件“/home/user/xmlpath/xmldocs.001.xml”，可以使用下列命令将数据导入回表中。

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

针对模式验证插入的文档

XMLVALIDATE 选项允许在导入 XML 文档时针对 XML 模式验证这些文档。在以下示例中，将针对导出 XML 文档时保存的模式信息验证入局 XML 文档:

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T1
```

指定解析选项

可以使用 XMLPARSE 选项来指定是保留还是去掉已导入的 XML 文档中的空格。在以下示例中，将针对导出 XML 文档时保存的 XML 模式信息验证已导入的所有 XML 文档，并在保留空格的情况下解析这些文档。

```
    IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
 WHITESPACE XMLVALIDATE USING XDS INSERT INTO USER.T1
```

装入 XML 数据

LOAD 实用程序可有效地将大量 XML 数据移到表中。

将数据装入到 XML 表列中时，可以使用 XML FROM 选项来指定一个或多个输入 XML 数据文件的路径。例如，要从 XML 文件 /home/user/xmlpath/xmlfile1.xml 中装入数据，应使用以下命令:

```
LOAD FROM data1.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

定界 ASCII 输入文件 data1.del 包含 XML 数据说明符（XDS），此 XML 数据说明符描述要装入的 XML 数据的位置。例如，以下 XDS 描述 xmldata.ext 文件中偏移量为 123 字节处的 XML 文档（其长度为 456 字节）:

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' />
```

支持使用已声明游标来装入 XML 数据。以下示例声明游标并使用该游标和 LOAD 命令将表 CUSTOMERS 中的数据添加到表 LEVEL1_CUSTOMERS 中:

```
DECLARE cursor_income_level1 CURSOR FOR
   SELECT * FROM customers
   WHERE XMLEXISTS('$DOC/customer[income_level=1]');

LOAD FROM cursor_income_level1 OF CURSOR INSERT INTO level1_customers;
```

将 XML 数据装入到 XML 列中时支持 LOAD 命令使用 ANYORDER 文件类型修饰符。

在分区数据库环境中装入 XML 数据

对于分布在数据库分区之间的表，可以并行方式将 XML 数据文件中的 XML 数据装入到表中。将文件中的 XML 数据装入到表中时，进行装入的所有数据库分区都必须可对 XML 数据文件进行读访问。

针对模式验证插入的文档

XMLVALIDATE 选项允许在装入 XML 文档时针对 XML 模式验证这些文档。在以下示例中，将针对由定界 ASCII 输入文件 data2.del 中的 XDS 标识的模式验证入局 XML 文档:

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T2
```

在这种情况下，XDS 包含 SCH 属性及用于验证的 XML 模式的标准 SQL 标识 “S1.SCHEMA_A”:

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' SCH='S1.SCHEMA_A' />
```

指定解析选项

可以使用 XMLPARSE 选项来指定是保留还是去掉装入的 XML 文档中的空格。在以下示例中，将针对带有 SQL 标识“S2.SCHEMA_A”的模式验证所有装入的 XML 文档，并且在保留空格的情况下解析这些文档：

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING SCHEMA S2.SCHEMA_A INSERT INTO USER.T1
```

解决装入 XML 数据时发生的建立索引错误

通过使用 db2diag 日志文件和 IMPORT 实用程序来标识并更正 XML 数据中的问题值，可以解决由于建立索引错误而失败的装入操作。

如果装入操作返回错误消息 SQL20305N（SQL 代码为 -20305），那么表示未能对一个或多个 XML 节点值建立索引。错误消息会输出该错误的原因码。在命令行处理器中输入 ? SQL20305N，以查找相应原因码的说明和用户响应。

对于执行插入操作期间产生的与建立索引有关的问题，生成的 XQuery 语句会输出到 db2diag 日志文件，以帮助在文档中找出失败的 XML 节点值。请参阅“常见 XML 建立索引问题”，以了解有关如何使用 XQuery 语句来查找失败的 XML 节点值的详细信息。

但是，对于执行装入操作期间产生的与建立索引有关的问题，生成的 XQuery 语句不会输出到 db2diag 日志文件。要生成这些 XQuery 语句，必须对未装入的失败行运行 IMPORT 实用程序。因为被拒绝的行不在表中，所以不能对失败文档运行 XQuery 语句。为解决此问题，必须创建带有相同定义的新表，但不包含任何索引。然后可将失败的行兰州到新表中，然后可对新表运行 XQuery 语句，以在文档中找出失败的 XML 节点值。

执行以下步骤来解决建立索引错误：

1. 使用输出信息中的记录号来确定装入操作期间被拒绝的行。
2. 创建仅包含被拒绝行的 .del 文件。
3. 创建包含原始表（T1）中的各列的新表（例如，T2）。不要对新表创建任何索引。
4. 将被拒绝的行装入到新表 T2 中。
5. 对于原始表 T1 中的每个被拒绝行：
   a. 将被拒绝的行导入 T1 以获取 SQL20305N 消息。导入会在遇到第一个错误时停止。
   b. 查看 db2diag 日志文件并获取生成的 XQuery 语句。要在输入文档中查找失败的节点值，请在 db2diag 日志文件中搜索字符串“SQL20305N”并与原因码编号匹配。依据原因码，您会找到一组指示信息和生成的 XQuery 语句，可使用它们来找出文档中导致该错误的问题值。
   c. 修改 XQuery 语句以使用新表 T2。
   d. 对 T2 运行 XQuery 语句以在文档中找出问题值。
   e. 在包含该文档的 .xml 文件中修正问题值。
   f. 返回步骤 A 并再次将被拒绝的行导入 T1。导致导入停止的行现在应该已成功插入。如果.del 文件还有一个被拒绝的行，那么 IMPORT 实用程序将在下一个错误处停止并再次输出 SQL20305N 消息。继续完成这些步骤直到导入运行成功。

示例

在以下示例中，已对 *date* 数据类型创建索引 BirthdateIndex。已指定 REJECT INVALID VALUES 选项，所以 */Person/Confidential/Birthdate* 的 XML 模式值必须全部对 *date* 数据类型有效。如果任何 XML 模式值不能转换为此数据类型，那么将返回错误。

使用下面的 XML 文档会装入 5 行，但第 1 行和第 4 行会被拒绝，原因是不能对 Birthdate 值建立索引。在文件 person1.xml 中，值 `March 16, 2002` 的日期格式不正确。在文件 person4.xml 中，值 `20000-12-09` 的年份部分存在多余的零，所以它是有效的 XML 日期值，但在 DB2 允许的年份范围（0001 至 9999）之外。已编辑一些样本输出以使示例更加精确。

要装入的 5 个 XML 文件如下所示:

person1.xml （Birthdate 值无效）

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>March 16, 2002</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person>
```

person2.xml（Birthdate 值有效）

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>2002-03-16</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person>
```

person3.xml（Birthdate 值有效）

```
<?xml version="1.0"?>
<Person gender="Female">
  <Name>
    <Last>McCarthy</Last>
    <First>Laura</First>
  </Name>
  <Confidential>
    <Age unit="years">6</Age>
    <Birthdate>2001-03-12</Birthdate>
    <SS>444-55-6666</SS>
  </Confidential>
  <Address>5960 Daffodil Lane, San Jose, CA 95120</Address>
</Person>
```

person4.xml（Birthdate 值无效）

```
<?xml version="1.0"?>
<Person gender="Female">
  <Name>
    <Last>Wong</Last>
    <First>Teresa</First>
  </Name>
  <Confidential>
    <Age unit="years">7</Age>
    <Birthdate>20000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential>
  <Address>5960 Tulip Court, San Jose, CA 95120</Address>
</Person>
```

person5.xml（Birthdate 值有效）

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Smith</Last>
    <First>Chris</First>
  </Name>
  <Confidential>
    <Age unit="years">10</Age>
    <Birthdate>1997-04-23</Birthdate>
    <SS>666-77-8888</SS>
  </Confidential>
  <Address>5960 Dahlia Street, San Jose, CA 95120</Address>
</Person>
```

输入文件 person.del 包含:

```
1, <XDS FIL='person1.xml'/>
2, <XDS FIL='person2.xml'/>
3, <XDS FIL='person3.xml'/>
4, <XDS FIL='person4.xml'/>
5, <XDS FIL='person5.xml'/>
```

DDL 和 LOAD 语句如下所示:

```
CREATE TABLE T1 (docID INT, XMLDoc XML);

CREATE INDEX BirthdateIndex ON T1(xmlDoc)
   GENERATE KEY USING XMLPATTERN '/Person/Confidential/Birthdate' AS SQL DATE
   REJECT INVALID VALUES;

LOAD FROM person.del OF DEL INSERT INTO T1
```

要解决尝试装入上述 XML 文件集合时发生的建立索引错误，应执行以下步骤:

1. 使用输出信息中的记录号来确定装入操作期间被拒绝的行。在以下输出中，记录号 1 和记录号 4 被拒绝。

   ```
   SQL20305N  由于在插入或更新表
   "LEECM.T1"中的"IID = 3"所标识的索引时检测到错误，无法插入或更新
   XML 值。原因码 ="5"。对于 XML 模式的相关原因码，
   XML 模式标识为"*N"，XML 模式数据类型为"*N"。 SQLSTATE=23525

   SQL3185W  处理输入文件中第"F0-1"行的数据时发生上述错误。

   SQL20305N  由于在插入或更新表
   "LEECM.T1"中的"IID = 3"所标识的索引时检测到错误，无法插入或更新
   XML 值。原因码 ="4"。对于 XML 模式的相关原因码，
   XML 模式标识为"*N"，XML 模式数据类型为"*N"。 SQLSTATE=23525

   SQL3185W  处理输入文件第"F0-4"行中的数据时发生了上述错误。
   ```

```
SQL3227W  记录标记"F0-1"指的是用户记录号"1"。

SQL3227W  记录标记"F0-4"指的是用户记录号"4"。

SQL3107W  消息文件中至少有一条警告消息。

   读取的行数         = 5
   跳过的行数         = 0
   装入的行数         = 3
   拒绝的行数         = 2
   删除的行数         = 0
   落实的行数         = 5
```

2. 使用被拒绝的行创建新文件 reject.del。

```
1, <XDS FIL='person1.xml'/>
4, <XDS FIL='person4.xml'/>
```

3. 创建包含原始表 T1 中的各列的新表 T2。不要对新表创建任何索引。

```
CREATE TABLE T2 LIKE T1
```

4. 将被拒绝的行装入到新表 T2 中。

```
LOAD FROM reject.del OF DEL INSERT INTO T2;
```

5. 对于原始表 T1 中的第 1 个被拒绝行:
   a. 将被拒绝的行导入 T1 以获取 -20305 消息。

   ```
   IMPORT FROM reject.del OF DEL INSERT INTO T1
   SQL3109N  实用程序开始从文件"reject.del"装入数据。

   SQL3306N  向表插入一行时，发生了 SQL 错误"-20305"。

   SQL20305N  不能插入或更新 XML 值，因为对表"LEECM.T1"插入或更新
   "IID = 3"标识的索引时检测到错误。

   原因码 ="5"。对于 XML 模式的相关原因码，
   XML 模式标识为"*N"，XML 模式数据类型为"*N"。
   SQLSTATE=23525

   SQL3110N  实用程序已完成处理。从输入文件读取了"1"行。
   ```

   b. 查看 db2diag 日志文件并获取生成的 XQuery 语句。

   ```
   函数: DB2 UDB，XML 存储器和索引管理器，xmlsDumpXQuery，探测: 608
   DATA #1: 字符串，36 字节
   SQL 代码: SQL20305N; 原因码: 5
   DATA #2: 字符串，265 字节
   要查找文档中导致错误的值，请使用一个 XML 列创建表，并在该表中插入失败的文档。
   在下面的查询中将表名和列名替换为新创建的表名和列名，并执行下列 XQuery。
   DATA #3: 字符串，247 字节
   xquery for $i in db2-fn:xmlcolumn(
     "LEECM.T1.XMLDOC")[/*:Person/*:Confidential/*:Birthdate="March 16, 2002"]
   return
   <Result>
    <ProblemDocument> {$i} </ProblemDocument>
    <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
   </Result>;
   ```

   c. 修改 XQuery 语句以使用新表 T2。

   ```
   xquery for $i in db2-fn:xmlcolumn(
     "LEECM.T2.XMLDOC")[/*:Person/*:Confidential/*:Birthdate="March 16, 2002"]
   return
   <Result>
    <ProblemDocument> {$i} </ProblemDocument>
    <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
   </Result>;
   ```

   d. 对表 T2 运行 XQuery 语句以查找文档中的问题值。

```
<Result><ProblemDocument><Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>March 16, 2002</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person></ProblemDocument><ProblemValue><Confidential>
    <Age unit="years">5</Age>
    <Birthdate>March 16, 2002</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential></ProblemValue></Result>
```

e. 在包含该文档的文件 person1.xml 中修正问题值。March 16, 2002 的日期格式不正确，所以它将更改为 2002-03-16。

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>2002-03-16</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person>
```

f. 返回步骤 A 以再次将被拒绝的行导入表 T1。

6. （步骤 5 的第一次重复）

a. 将被拒绝的行导入表 T1。第 1 行现在已成功导入，原因是从导入文件中读取了两行。第 2 行出现了新错误。

```
IMPORT FROM reject.del OF DEL INSERT INTO T1
SQL3109N  实用程序开始从文件"reject.del"装入数据。

SQL3306N  向表插入一行时，发生了 SQL 错误"-20305"。

SQL20305N  不能插入或更新 XML 值，因为对表"LEECM.T1"插入或更新
"IID = 3"标识的索引时检测到错误。

原因码 ="4"。对于 XML 模式的相关原因码，
XML 模式标识为"*N"，XML 模式数据类型为"*N"。
SQLSTATE=23525

SQL3110N  实用程序已完成处理。从输入文件读取了"2"行。
```

b. 查看 db2diag 日志文件并获取生成的 XQuery 语句。

```
函数：DB2 UDB，XML 存储器和索引管理器，xmlsDumpXQuery，探测：608
DATA #1：字符串，36 字节
SQL 代码：SQL20305N；原因码：4
DATA #2：字符串，265 字节
要查找文档中导致错误的值，请使用一个 XML 列创建表，并在该表中插入失败的文档。
在下面的查询中将表名和列名替换为新创建的表名和列名，并执行下列 XQuery。
DATA #3：字符串，244 字节
xquery for $i in db2-fn:xmlcolumn("LEECM.T1.XMLDOC")
   [/*:Person/*:Confidential/*:Birthdate="20000-12-09"]
return
```

```
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
</Result>;
```

c. 修改 XQuery 语句以使用表 T2。

```
xquery for $i in db2-fn:xmlcolumn("LEECM.T2.XMLDOC")
   [/*:Person/*:Confidential/*:Birthdate="20000-12-09"]
return
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
</Result>;
```

d. 运行 XQuery 语句以查找文档中的问题值。

```
<Result><ProblemDocument><Person gender="Female">
  <Name>
    <Last>Wong</Last>
    <First>Teresa</First>
  </Name>
  <Confidential>
    <Age unit="years">7</Age>
    <Birthdate>20000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential>
  <Address>5960 Tulip Court, San Jose, CA 95120</Address>
</Person></ProblemDocument><ProblemValue><Confidential>
    <Age unit="years">7</Age>
    <Birthdate>20000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential></ProblemValue></Result>
```

e. 在包含该文档的文件 person4.xml 中修正问题值。值 20000-12-09 的年份部分存在多余的零，所以它在 DB2 允许的年份范围（0001 至 9999）之外。该值将更改为 2000-12-09。

```
<?xml version="1.0"?>
<Person gender="Female">
  <Name>
    <Last>Wong</Last>
    <First>Teresa</First>
  </Name>
  <Confidential>
    <Age unit="years">7</Age>
    <Birthdate>2000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential>
  <Address>5960 Tulip Court, San Jose, CA 95120</Address>
</Person>
```

f. 返回步骤 A 以再次将被拒绝的行导入 T1。

7. （步骤 5 的第二次重复）

a. 将被拒绝的行导入 T1。

```
IMPORT FROM reject.del OF DEL INSERT INTO T1
SQL3109N  实用程序开始从文件"reject.del"装入数据。

SQL3110N  实用程序已完成处理。从输入文件读取了"2"行。

SQL3221W  ...开始 COMMIT WORK。输入记录计数 ="2"。

SQL3222W  ...COMMIT 任何数据库更改成功。

SQL3149N  处理了输入文件中的"2"行。已在表中成功插入了"2"行。"0"行被拒绝。

   读取的行数          = 2
```

```
跳过的行数        = 0
插入的行数        = 2
更新的行数        = 0
拒绝的行数        = 0
落实的行数        = 2
```

现在已解决该问题。person.del 的所有行已成功插入到表 T1 中。

第 10 章 应用程序编程语言支持

可以编写应用程序来将 XML 数据存储在 DB2 数据库表中，从表中检索数据或调用具有 XML 参数的存储过程或用户定义的函数。

可以使用下列任何语言来编写应用程序:
- C 或 C++（嵌入式 SQL 或 DB2 CLI）
- COBOL
- Java（JDBC 或 SQLJ）
- C# 和 Visual Basic（IBM 数据服务器 .NET 提供程序）
- PHP
- Perl

应用程序可以从 XML 列中检索整个文档或文档片段。但是，只能将整个文档存储在 XML 列中。

存储过程和用户定义的函数可以在输入或输出参数中传递 XML 值。将 XML 数据作为 IN、OUT 或 INOUT 参数传递至存储过程时，将具体化该数据。如果使用的是 Java 存储过程，那么可能需要根据 XML 参数的数量和大小以及正在同时执行的外部存储过程的数目来增大堆大小（JAVA_HEAP_SZ 配置参数）。要调用具有 XML 或 XML AS CLOB 参数的存储过程或用户定义的函数，使用兼容数据类型执行 CALL 语句。

应用程序为 DB2 数据库服务器提供 XML 值时，该数据库服务器会将数据从序列化的 XML 字符串格式转换为使用 Unicode UTF-8 编码的 XML 分层格式。

应用程序从 XML 列中检索数据时，DB2 数据库服务器将数据从 XML 分层格式转换为序列化的 XML 字符串格式。此外，数据库服务器可能需要将输出从 UTF-8 转换为应用程序编码。

检索 XML 数据时，需要了解代码页转换对数据丢失的影响。当目标代码页中无法表示源代码页中的字符时，就会丢失数据。

应用程序可以从 XML 列中检索整个 XML 文档或一个序列。

访存整个 XML 文档时，将该文档检索到应用程序变量中。

检索 XML 序列时，您有几种选择:
- 直接执行 XQuery 表达式。

  要在应用程序中执行 XQuery 表达式，先将字符串 'XQUERY' 附加到 XQuery 表达式之前，然后动态执行生成的字符串。

  直接执行 XQuery 表达式时，DB2 数据库服务器返回 XQuery 语句的结果序列作为结果表。结果表中的每行是序列中的一项。
- 在 SQL SELECT 或单行 SELECT INTO 操作内，调用 XMLQUERY 或 XMLTABLE 内置函数并将 XQuery 表达式作为自变量传递。

此方法可与静态或动态 SQL 以及任何应用程序编程语言配合使用。XMLQUERY 是一个标量函数，它返回应用程序变量中的整个序列。XMLTABLE 是一个表函数，它返回序列中的每一项作为结果表的行。结果表中的列是检索到的序列项中的值。

参数标记和主变量

不能在 XQuery 表达式中的任何位置（包括在 XQuery 表达式中指定的 SQL 中）指定参数标记或主变量。例如，XQuery 函数 db2-fn:sqlquery 允许您指定一个具有 XQuery 表达式的 SQL 全查询，来抽取对产品的详细描述:

```
xquery
db2-fn:sqlquery("select description from product where pid='100-103-01'")
                /product/description/details/text()
```

不能在 XQuery 表达式中指定参数标记或主变量,即使在全查询中也是如此。以下表达式是错误的，也不受支持（它将返回 SQLSTATE 42610，sqlcode -418）:

```
xquery
db2-fn:sqlquery("select description from product where pid=?")
                /product/description/details/text()
```

要将应用程序值传递给 XQuery 表达式，可使用 SQL/XML 函数 XMLQUERY 和 XMLTABLE。这些函数的 PASSING 子句允许您在对 XQuery 表达式求值期间使用应用程序值。

以下查询说明如何使用 SQL/XML 来重写先前的错误查询，以便获得等价的结果:

```
SELECT XMLQUERY ('$descdoc/product/description/details/text()'
   passing description as "descdoc")
FROM product
WHERE pid=?
```

CLI

CLI 应用程序中的 XML 数据处理 － 概述

DB2 CLI 应用程序可以检索和存储使用 SQL_XML 数据类型的 XML 数据。此数据类型对应于 DB2 数据库的本机 XML 数据类型，该数据类型用来定义用于存储结构良好的 XML 文档的列。可以将 SQL_XML 类型绑定至下列 C 类型: SQL_C_BINARY、SQL_C_CHAR、SQL_C_WCHAR 和 SQL_C_DBCHAR。但是，建议您使用缺省 SQL_C_BINARY 类型而不使用字符类型，这样可以避免在使用字符类型时因执行代码页转换而可能导致丢失或破坏数据。

要将 XML 数据存储在 XML 列中，应将包含 XML 值的二进制（SQL_C_BINARY）或字符（SQL_C_CHAR、SQL_C_WCHAR 或 SQL_C_DBCHAR）缓冲区绑定至 SQL_XML SQL 类型，然后执行 INSERT 或 UPDATE SQL 语句。要从数据库中检索 XML 数据，应将结果集绑定至二进制（SQL_C_BINARY）或字符（SQL_C_CHAR、SQL_C_WCHAR 或 SQL_C_DBCHAR）类型。由于存在编码问题，因此应谨慎使用字符类型。

将 XML 值检索到应用程序数据缓冲区之后，DB2 服务器将对 XML 值执行隐式序列化，以便将它从已存储的分层格式转换为已序列化的字符串格式。对于字符类型的缓冲区，XML 值将被隐式序列化为与字符类型相关联的应用程序字符代码页。

缺省情况下，XML 声明包含在输出序列化字符串中。可以通过设置 SQL_ATTR_XML_DECLARATION 语句或连接属性，或者通过在 db2cli.ini 文件中设置 XMLDeclaration CLI/ODBC 配置关键字来更改此缺省行为。

可以在 DB2 CLI 应用程序中发出并执行 XQuery 表达式和 SQL/XML 函数。像任何其他 SQL 语句一样发出并执行 SQL/XML 函数。必须在 XQuery 表达式前面添加不区分大小写的关键字“XQUERY”，或者必须为与 XQuery 表达式相关联的语句句柄设置 SQL_ATTR_XQUERY_STATEMENT 语句属性。

CLI 应用程序中的 XML 列插入和更新

当更新一个表的 XML 列或者向这些列中插入数据时，输入数据必须采用已序列化的字符串格式。

对于 XML 数据，当您使用 SQLBindParameter() 来将参数标记绑定至输入数据缓冲区时，可以将输入数据缓冲区的数据类型指定为 SQL_C_BINARY、SQL_C_CHAR、SQL_C_DBCHAR 或 SQL_C_WCHAR。

当您将包含 XML 数据的数据缓冲区作为 SQL_C_BINARY 来绑定时，DB2 CLI 会将该 XML 数据作为内部编码的数据来处理。这是首选方法，因为它避免了在使用字符类型时进行字符转换所需要的开销和可能产生的数据丢失。

重要事项: 如果 XML 数据是采用应用程序代码页编码方案之外的编码方案和 CCSID 来进行编码的，那么需要在该数据中包含内部编码，并将数据作为 SQL_C_BINARY 来绑定以避免进行字符转换。

当您将包含 XML 数据的数据缓冲区作为 SQL_C_CHAR、SQL_C_DBCHAR 或 SQL_C_WCHAR 来绑定时，DB2 CLI 会将 XML 数据作为外部编码的数据来处理。DB2 CLI 按如下所示来确定数据的编码:

- 如果 C 类型为 SQL_C_WCHAR，那么 DB2 CLI 将假定数据是采用 UCS-2 来编码的。
- 如果 C 类型为 SQL_C_CHAR 或 SQL_C_DBCHAR，那么 DB2 CLI 将假定数据是采用应用程序代码页编码方案来编码的。

如果需要数据库服务器在将数据存储到 XML 列中之前隐式解析该数据，那么应将 SQLBindParameter() 中的参数标记数据类型指定为 SQL_XML。

建议进行隐式解析，因为使用 XMLPARSE 来显式解析字符类型将产生编码问题。

以下示例说明如何使用建议的 SQL_C_BINARY 类型来更新 XML 列中的 XML 数据。

```
char xmlBuffer[10240];
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

// xmlBuffer contains an internally encoded XML document that is to replace
// the existing XML document
length = strlen (xmlBuffer);
SQLPrepare (hStmt, "UPDATE dept SET deptdoc = ? WHERE id = '001'", SQL_NTS);
WHERE id = '001'", SQL_NTS);
SQLBindParameter (hStmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_XML, 0, 0,
                                  xmlBuffer, 10240, &length);
SQLExecute (hStmt);
```

在 CLI 应用程序中检索 XML 数据

当从表的 XML 列中选择数据时，输出数据采用已序列化的字符串格式。

对于 XML 数据，当您使用 SQLBindCol() 来将查询结果集中的列绑定至应用程序变量时，可以将应用程序变量的数据类型指定为 SQL_C_BINARY、SQL_C_CHAR、SQL_C_DBCHAR 或 SQL_C_WCHAR。当从 XML 列中检索结果集时，建议您将应用程序变量绑定至 SQL_C_BINARY 类型。如果绑定至字符类型，那么可能会因执行代码页转换而造成数据丢失。当目标代码页中无法表示源代码页中的字符时，就会丢失数据。而将变量绑定至 SQL_C_BINARY C 类型就可以避免这些问题。

XML 数据将作为内部编码的数据返回给应用程序。DB2 CLI 按如下所示来确定数据的编码:

- 如果 C 类型为 SQL_C_BINARY，那么 DB2 CLI 将采用 UTF-8 编码方案来返回数据。
- 如果 C 类型为 SQL_C_CHAR 或 SQL_C_DBCHAR，那么 DB2 CLI 将采用应用程序代码页编码方案来返回数据。
- 如果 C 类型为 SQL_C_WCHAR，那么 DB2 CLI 将采用 UCS-2 编码方案来返回数据。

在将数据返回给应用程序之前，数据库服务器将对数据执行隐式序列化。可以通过调用 XMLSERIALIZE 函数来将 XML 数据显式序列化为特定数据类型。但是，建议您使用隐式序列化，因为使用 XMLSERIALIZE 来对字符类型进行显式序列化时可能会产生编码问题。

以下示例说明如何将 XML 列中的 XML 数据检索到二进制应用程序变量中。

```
char xmlBuffer[10240];
// xmlBuffer is used to hold the retrieved XML document
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

length = sizeof (xmlBuffer);
SQLExecute (hStmt, "SELECT deptdoc FROM dept WHERE id='001'", SQL_NTS);
SQLBindCol (hStmt, 1, SQL_C_BINARY, xmlBuffer, &length, NULL);
SQLFetch (hStmt);
SQLCloseCursor (hStmt);
// xmlBuffer now contains a valid XML document encoded in UTF-8
```

更改 CLI 应用程序中的缺省 XML 类型处理

DB2 CLI 支持 CLI/ODBC 配置关键字，对于在描述或指定 XML 列和参数标记的 SQL_C_DEFAULT 时不期望返回缺省类型的应用程序，这些配置关键字将提供兼容性。当描述 XML 列或参数时，较旧的 CLI 和 ODBC 应用程序可能不识别或者不期望缺省 SQL_XML 类型。对于 XML 列和参数标记，某些 CLI 或 ODBC 应用程序可能还期望 SQL_C_BINARY 之外的缺省类型。为了对这些类型的应用程序提供兼容性，DB2 CLI 支持 MapXMLDescribe 和 MapXMLCDefault 关键字。

MapXMLDescribe 指定在描述 XML 列或参数标记时将返回哪种 SQL 数据类型。

MapXMLCDefault 指定在为 DB2 CLI 函数中的 XML 列和参数标记指定 SQL_C_DEFAULT 时将使用 C 类型。

嵌入式 SQL

在嵌入式 SQL 应用程序中声明 XML 主变量

要实现在数据库服务器与嵌入式 SQL 应用程序之间交换 XML 数据，需要在应用程序源代码中声明主变量。

DB2 V9.1 引入了一种 XML 数据类型，该数据类型将 XML 数据存储在一组采用树形结构的结构化节点中。具有此 XML 数据类型的列被描述为 SQL_TYP_XML 列 SQLTYPE，并且应用程序可以对这些列或参数的输入和输出绑定各种特定于语言的数据类型。可以直接使用 SQL、SQL/XML 扩展或 XQuery 来访问 XML 列。XML 数据类型不只是适用于列。函数可以将 XML 值用作自变量，还可以生成 XML 值。同样地，存储过程可以采用 XML 值作为输入参数和输出参数。最后，无论 XQuery 表达式是否访问 XML 列，它们都会生成 XML 值。

从本质上来说，XML 数据是字符，并且具有用来指定所使用的字符集的编码。可以在外部确定 XML 数据的编码，它是从包含 XML 文档的序列化字符串表示的基本应用程序类型派生而来的。也可以在内部确定它，但是需要对数据进行解释。对于 Unicode 编码文档，建议使用字节顺序标记（BOM），它由位于数据流开头的 Unicode 字符代码组成。BOM 用作一个特征符，它定义字节顺序和 Unicode 编码格式。

除了 XML 主变量以外，还可以使用现有字符和二进制类型（包括 CHAR、VARCHAR、CLOB 和 BLOB）来访存和插入数据。但是，它们不会像 XML 主变量那样依赖于 XML 解析。而是引入并应用了具有缺省空格去掉功能的显式 XMLPARSE 函数。

有关开发嵌入式 SQL 应用程序的 XML 和 XQuery 限制

要在嵌入式 SQL 应用程序中声明 XML 主变量:

在应用程序的声明部分，将 XML 主变量声明为 LOB 数据类型:

-
  ```
  SQL TYPE IS XML AS CLOB(n) <hostvar_name>
  ```
  其中 <hostvar_name> 是一个 CLOB 主变量，它包含使用应用程序的混合代码页编码的 XML 数据。
-
  ```
  SQL TYPE IS XML AS DBCLOB(n) <hostvar_name>
  ```
  其中 <hostvar_name> 是一个 DBCLOB 主变量，它包含使用应用程序图形代码页编码的 XML 数据。
-
  ```
  SQL TYPE IS XML AS BLOB(n) <hostvar_name>
  ```
  其中 <hostvar_name> 是一个 BLOB 主变量，它包含在内部编码的 XML 数据[1]。
-
  ```
  SQL TYPE IS XML AS CLOB_FILE <hostvar_name>
  ```
  其中 <hostvar_name> 是一个 CLOB 文件，它包含使用应用程序混合代码页编码的 XML 数据。

-

  ```
  SQL TYPE IS XML AS DBCLOB_FILE <hostvar_name>
  ```

  其中 <hostvar_name> 是一个 DBCLOB 文件，它包含使用应用程序图形代码页编码的 XML 数据。

-

  ```
  SQL TYPE IS XML AS BLOB_FILE <hostvar_name>
  ```

  其中 <hostvar_name> 是一个 BLOB 文件，它包含在内部编码的 XML 数据[1]。

注:

1. 请参阅用于根据 XML 1.0 规范确定编码的算法（http://www.w3.org/TR/REC-xml/#sec-guessing-no-ext-info）。

示例：引用嵌入式 SQL 应用程序中的 XML 主变量

以下样本应用程序说明了如何引用使用 C 和 COBOL 语言的 XML 主变量。

示例：嵌入式 SQL C 应用程序

以下代码示例进行了格式编排以便更加清楚明了：

```
EXEC SQL BEGIN DECLARE;
    SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
    SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
    SQL TYPE IS CLOB( 10K ) clobBuf;
  EXEC SQL END DECLARE SECTION;

// as XML AS CLOB
// The XML value written to xmlBuf will be prefixed by an XML declaration
// similar to: <?xml version = "1.0" encoding = "ISO-8859-1" ?>
// Note: The encoding name will depend upon the application codepage
EXEC SQL SELECT xmlCol INTO :xmlBuf
      FROM myTable
      WHERE id = '001'; EXEC SQL UPDATE myTable
      SET xmlCol = :xmlBuf
      WHERE id = '001';
// as XML AS BLOB
// The XML value written to xmlblob will be prefixed by an XML declaration
// similar to: <?xml version = "1.0" encoding = "UTF-8"?>
EXEC SQL SELECT xmlCol INTO :xmlblob
      FROM myTable
      WHERE id = '001'; EXEC SQL UPDATE myTable
      SET xmlCol = :xmlblob
      WHERE id = '001';
// as CLOB
// The output will be encoded in the application character codepage,
// but will not contain an XML declaration
EXEC SQL SELECT XMLSERIALIZE (xmlCol AS CLOB(10K)) INTO :clobBuf
      FROM myTable
      WHERE id = '001'; EXEC SQL UPDATE myTable
      SET xmlCol = XMLPARSE (:clobBuf PRESERVE WHITESPACE)
      WHERE id = '001';
```

示例：嵌入式 SQL COBOL 应用程序

以下代码示例进行了格式编排以便更加清楚明了：

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
    01 xmlBuf USAGE IS SQL TYPE IS XML as CLOB(5K).
    01 clobBuf USAGE IS SQL TYPE IS CLOB(5K).
    01 xmlblob  USAGE IS SQL TYPE IS BLOB(5K).
EXEC SQL END DECLARE SECTION END-EXEC.
```

```
* as XML
EXEC SQL SELECT xmlCol INTO :xmlBuf
      FROM myTable
   WHERE id = '001' END-EXEC.
EXEC SQL UPDATE myTable
      SET xmlCol = :xmlBuf
   WHERE id = '001' END-EXEC.

* as BLOB
EXEC SQL SELECT xmlCol INTO :xmlblob
      FROM myTable
   WHERE id = '001' END-EXEC.
EXEC SQL UPDATE myTable
      SET xmlCol = :xmlblob
   WHERE id = '001' END-EXEC.

* as CLOB
EXEC SQL SELECT XMLSERIALIZE(xmlCol AS CLOB(10K)) INTO :clobBuf
      FROM myTable
   WHERE id= '001' END-EXEC.
EXEC SQL UPDATE myTable
      SET xmlCol = XMLPARSE(:clobBuf) PRESERVE WHITESPACE
   WHERE id = '001' END-EXEC.
```

执行嵌入式 SQL 应用程序中的 XQuery 表达式

可以将 XML 数据存储在表中，还可以使用嵌入式 SQL 应用程序并使用 XQuery 表达式来访问 XML 列。要访问 XML 数据，可使用 XML 主变量，而不是将数据强制转换为字符或二进制数据类型。如果您不利用 XML 主变量，那么用于访问 XML 数据的最佳替代方法是使用 FOR BIT DATA 或 BLOB 数据类型，以避免进行代码页转换。

- 在嵌入式 SQL 应用程序中声明 XML 主变量。
- 必须使用 XML 类型来检索静态 SQL `SELECT INTO` 语句中的 XML 值。
- 如果对需要 XML 值的输入使用 CHAR、VARCHAR、CLOB 或 BLOB 主变量，那么值将依赖于具有缺省空格（STRIP）处理的 XMLPARSE 函数操作。否则，需要使用 XML 主变量。

要直接发出嵌入式 SQL 应用程序中的 XQuery 表达式，应在表达式前面添加“XQUERY”关键字。对于静态 SQL，使用 XMLQUERY 函数。当调用 XMLQUERY 函数时，不会在 XQuery 表达式前面添加“XQUERY”。

这些示例将返回样本数据库的 CUSTOMER 表中 XML 文档中的数据。

示例 1：通过在前面添加“XQUERY”关键字来直接执行 C 和 C++ 动态 SQL 中的 XQuery 表达式

在 C 和 C++ 应用程序中，可以采用以下方式来发出 XQuery 表达式:

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  char stmt[16384];
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
EXEC SQL END DECLARE SECTION;

sprintf( stmt, "XQUERY (for $a in db2-fn:xmlcolumn("CUSTOMER.INFO")
        /*:customerinfo[*:addr/*:city = "Toronto"]/@Cid return data($a))");

    EXEC SQL PREPARE s1 FROM :stmt;
    EXEC SQL DECLARE c1 CURSOR FOR s1;
    EXEC SQL OPEN c1;
```

```
    while( sqlca.sqlcode == SQL_RC_OK )
{
        EXEC SQL FETCH c1 INTO :xmlblob;
        /* Display results */
}

    EXEC SQL CLOSE c1;
    EXEC SQL COMMIT;
```

示例 2: 执行使用 XMLQUERY 函数和 XMLEXISTS 谓词的静态 SQL 中的 XQuery 表达式

可以按如下所示静态编译包含 XMLQUERY 函数的 SQL 语句:

```
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
  EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE C1 CURSOR FOR SELECT XMLQUERY(data($INFO/*:customerinfo/@Cid)')
FROM customer
WHERE XMLEXISTS('$INFO/*:customerinfo[*:addr/*:city = "Toronto"]');

EXEC SQL OPEN c1;

while( sqlca.sqlcode == SQL_RC_OK )
{
  EXEC SQL FETCH c1 INTO :xmlblob;
  /* Display results */
}

EXEC SQL CLOSE c1;
EXEC SQL COMMIT;
```

示例 3: 执行 COBOL 嵌入式 SQL 应用程序中的 XQuery 表达式

在 COBOL 应用程序中，可以采用以下方式来发出 XQuery 表达式:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
              01 stmt pic x(80).
       01 xmlBuff USAGE IS SQL TYPE IS XML AS BLOB (10K).
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE "XQUERY (for $a in db2-fn:xmlcolumn("CUSTOMER.INFO")/*:customerinfo
             [*:addr/*:city = "Toronto"]/@Cid return data($a)))" TO stmt.
    EXEC SQL PREPARE s1 FROM :stmt END-EXEC.
    EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.
    EXEC SQL OPEN c1 USING :host-var END-EXEC.

    *Call the FETCH and UPDATE loop.
    Perform Fetch-Loop through End-Fetch-Loop
          until SQLCODE does not equal 0.

    EXEC SQL CLOSE c1 END-EXEC.
    EXEC SQL COMMIT END-EXEC.

    Fetch-Loop Section.
            EXEC SQL FETCH c1 INTO :xmlBuff END-EXEC.
            if SQLCODE not equal 0
                  go to End-Fetch-Loop.
    * Display results
    End-Fetch-Loop. exit.
```

关于使用 XML 和 XQuery 开发嵌入式 SQL 应用程序的建议

对于在嵌入式 SQL 应用程序中使用 XML 和 XQuery，提供了下列建议，并且还存在下列限制。

- 应用程序必须访问采用已序列化字符串格式的所有 XML 数据。
  - 必须采用已序列化字符串格式来表示所有数据（包括数字和日期时间数据）。
- 外部化的 XML 数据最多只能为 2 GB。
- 包含 XML 数据的所有游标都是非分块的（每个访存操作都将生成一个数据库服务器请求）。
- 每当字符主变量中包含已序列化的 XML 数据时，就假定将应用程序代码页用作数据的编码，并且必须与数据中存在的任何内部编码相匹配。
- 必须将 LOB 数据类型指定为 XML 主变量的基本类型。
- 下列限制适用于静态 SQL:
  - 不能使用字符和二进制主变量来从 `SELECT INTO` 操作中检索 XML 值。
  - 在输入需要 XML 数据类型的情况下，使用 CHAR、VARCHAR、CLOB 和 BLOB 主变量将依赖于具有缺省空格处理特征 (`'STRIP WHITESPACE'`) 的 `XMLPARSE` 操作。而任何其他非 XML 主变量类型都将被拒绝。
  - 不支持静态 XQuery 表达式；尝试预编译 XQuery 表达式时将失败，并且会产生错误。只能通过 XMLQUERY 函数来发出 XQuery 表达式。
- 通过在 XQuery 表达式前面添加一个“XQUERY”字符串，就可以动态发出该表达式。

标识 SQLDA 中的 XML 值

要指示基本类型具有 XML 数据，必须按如下所示更新 SQLVAR 的 sqlname 字段:

- `sqlname.length` 必须为 8
- sqlname.data 的前两个字节必须是 `X'0000'`
- sqlname.data 的第三和第四个字节必须是 `X'0000'`
- `sqlname.data` 的第五个字节必须是 `X'01'`（仅当满足前两个条件时才称为 XML 子类型指示符）
- 其余字节必须是 `X'000000'`

如果在 SQLTYPE 不是 LOB 的 SQLVAR 中设置了 XML 子类型指示符，那么在运行时将返回 SQL0804 错误（rc=115）。

注: 只能从 DESCRIBE 语句中返回 SQL_TYP_XML。此类型不能用于任何其他请求。应用程序必须修改 SQLDA 以包含有效字符或二进制类型，并适当地设置 `sqlname` 字段以指示数据是 XML。

Java

JDBC

JDBC 应用程序中的 XML 数据

在 JDBC 应用程序中，可以将数据存储在 XML 列中并从 XML 列中检索数据。

在数据库表中，XML 内置数据类型用于将 XML 数据以一组采用树形结构的结构化节点形式存储在列中。

在应用程序中，XML 数据采用已序列化字符串格式。

在连接至 DB2 数据库 Linux 版、UNIX 版和 Windows 版的 JDBC 应用程序中，XML 数据采用文本 XML 格式。在连接至 DB2 z/OS 版的 JDBC 应用程序中，XML 数据可以采用文本 XML 格式或者二进制 XML 格式。

在 JDBC 应用程序中，您可以执行下列操作:

- 使用 setXXX 方法将整个 XML 文档存储在一个 XML 列中。
- 使用 getXXX 方法从一个 XML 列中检索整个 XML 文档。
- 通过以下方法从 XML 列中的文档内检索一个序列：使用 SQL XMLQUERY 函数将该序列检索到数据库中的一个已序列化序列中，然后使用 getXXX 方法将数据检索到应用程序变量中。
- 通过以下方法从 XML 列中的文档内检索一个序列：使用前面添加了字符串“XQUERY”的 XQuery 表达式来将该序列的元素检索到数据库中的结果表中，而结果表中的每一行都表示该序列中的一项。然后使用 getXXX 方法将数据检索到应用程序变量中。
- 通过以下方法从 XML 列中的文档内检索一个作为用户定义的表的序列：使用 SQL XMLTABLE 函数来定义并检索结果表。然后使用 getXXX 方法来将数据从结果表检索到应用程序变量中。

可以使用 JDBC 4.0 `java.sql.SQLXML` 对象来检索和更新 XML 列中的数据。如果调用诸如 ResultSetMetaData.getColumnTypeName 之类的元数据方法，那么会对 XML 列类型返回整数值 `java.sql.Types.SQLXML`。

JDBC 应用程序中的 XML 列更新

当您更新数据库表的 XML 列或者向这些列中插入数据时，JDBC 应用程序中的输入数据必须采用已序列化的字符串格式。

下表列示了可用来将数据放入 XML 列的方法和相应的输入数据类型。

表 32. 用于更新 XML 列的方法和数据类型

方法	输入数据类型
PreparedStatement.setAsciiStream	InputStream
PreparedStatement.setBinaryStream	InputStream
PreparedStatement.setBlob	Blob
PreparedStatement.setBytes	byte[]
PreparedStatement.setCharacterStream	Reader
PreparedStatement.setClob	Clob
PreparedStatement.setObject	byte[]、Blob、Clob、SQLXML、DB2Xml（不推荐使用）、InputStream、Reader 和 String
PreparedStatement.setSQLXML	SQLXML
PreparedStatement.setString	String

XML 数据的编码可以从数据本身派生（称为*内部编码*数据），也可以从外部源派生（称为*外部编码*数据）。作为二进制数据发送至数据库服务器的 XML 数据被当作内部编码的数据来处理。作为字符数据发送至数据源的 XML 数据被当作外部编码的数据来处理。

Java 应用程序的外部编码始终为 Unicode 编码。

外部编码的数据可以具有内部编码。即，可以将数据作为字符数据发送至数据源，但该数据中包含编码信息。数据源按如下所示来处理内部编码与外部编码之间的不兼容性:

- 如果数据源是 DB2 数据库 Linux 版、UNIX 版和 Windows 版，那么在外部编码与内部编码不兼容的情况下，数据库源将产生错误，除非外部编码和内部编码都是 Unicode。如果外部编码和内部编码都是 Unicode，那么数据库源将忽略内部编码。
- 如果数据库源是 DB2 z/OS 版，那么它将忽略内部编码。

XML 列中的数据使用 UTF-8 编码来存储。数据库源负责处理将数据从内部编码或外部编码转换为 UTF-8。

示例: 以下示例说明了如何将 SQLXML 对象中的数据插入 XML 列中。数据是字符串数据，因此，数据库源将它们当作外部编码来处理。

```
  public void insertSQLXML()
  {
   Connection con = DriverManager.getConnection(url);
    SQLXML info = con.createSQLXML();
                            // Create an SQLXML object
    PreparedStatement insertStmt = null;
    String infoData =
      "<customerinfo xmlns=""http://posample.org"" " +
      "Cid=""1000"">...</customerinfo>";
    info.setString(infoData);
                            // Populate the SQLXML object
    int cid = 1000;
        try {
            sqls = "INSERT INTO CUSTOMER (CID, INFO) VALUES (?, ?)";
insertStmt = con.prepareStatement(sqls);
      insertStmt.setInt(1, cid);
      insertStmt.setSQLXML(2, info);
                            // Assign the SQLXML object value
                 // to an input parameter
            if (insertStmt.executeUpdate() != 1) {
        System.out.println("insertSQLXML: No record inserted.");
      }
    }
        catch (IOException ioe) {
          ioe.printStackTrace();
    }
        catch (SQLException sqle) {
      System.out.println("insertSQLXML: SQL Exception: " +
                sqle.getMessage());
      System.out.println("insertSQLXML: SQL State: " +
                sqle.getSQLState());
      System.out.println("insertSQLXML: SQL Error Code: " +
                sqle.getErrorCode());

    }
  }
```

示例: 以下示例说明了如何将文件中的数据插入 XML 列中。数据是作为二进制数据插入的，因此，数据库服务器支持内部编码。

```
  public void insertBinStream(Connection conn)
  {
    PreparedStatement insertStmt = null;
    String sqls = null;
        int cid = 0;
        Statement stmt=null;
        try {
            sqls = "INSERT INTO CUSTOMER (CID, INFO) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
      insertStmt.setInt(1, cid);
            File file = new File(fn);
            insertStmt.setBinaryStream(2,
        new FileInputStream(file), (int)file.length());
      if (insertStmt.executeUpdate() != 1) {
                System.out.println("insertBinStream: No record inserted.");
}
    }
        catch (IOException ioe) {
          ioe.printStackTrace();
    }
        catch (SQLException sqle) {
            System.out.println("insertBinStream: SQL Exception: " +
                sqle.getMessage());
            System.out.println("insertBinStream: SQL State: " +
                sqle.getSQLState());
            System.out.println("insertBinStream: SQL Error Code: " +
                sqle.getErrorCode());

    }
  }
```

在 JDBC 应用程序中检索 XML 数据

在 JDBC 应用程序中，可以使用 ResultSet.getXXX 或 ResultSet.getObject 方法从 XML 列中检索数据。

从 DB2 表的 XML 列中检索数据时，输出数据采用已序列化的字符串格式。无论您是检索 XML 列的整个内容还是从该列中检索某一序列，情况都是如此。

可以使用下列其中一种技术来检索 XML 数据:

- 使用 ResultSet.getSQLXML 方法来检索数据。然后，使用 SQLXML.getXXX 方法将数据检索到兼容的输出数据类型中。此技术需要使用 JDBC 4.0 或更高版本。
- 使用 ResultSet.getXXX 方法而不是 ResultSet.getObject 来将数据检索到兼容的数据类型中。
- 使用 ResultSet.getObject 方法来检索数据，然后将它的数据类型强制转换为 DB2Xml 类型，并将它指定给 DB2Xml 对象。然后使用 DB2Xml.getDB2XXX 或 DB2Xml.getDB2XmlXXX 方法来将数据检索到兼容的输出数据类型中。

  此技术使用已不推荐使用的 DB2Xml 对象。首选使用先前描述的技术。

下表列示了用于检索 XML 数据的 ResultSet 方法和相应的输出数据类型。

表 33. 用于检索 XML 数据的 ResultSet 方法和数据类型

方法	输出数据类型
ResultSet.getAsciiStream	InputStream
ResultSet.getBinaryStream	InputStream
ResultSet.getBytes	byte[]

表 33. 用于检索 XML 数据的 ResultSet 方法和数据类型 (续)

方法	输出数据类型
ResultSet.getCharacterStream	Reader
ResultSet.getObject	Object
ResultSet.getSQLXML	SQLXML
ResultSet.getString	String

下表列示了一些方法，可以通过调用这些方法来从 java.sql.SQLXML 或 com.ibm.db2.jcc.DB2Xml 对象中检索数据，以及检索相应的输出数据类型和 XML 声明中的编码类型。

表 34. SQLXML 和 DB2Xml 方法、数据类型以及添加的编码规范

方法	输出数据类型	已添加的 XML 内部编码声明的类型
SQLXML.getBinaryStream	InputStream	无
SQLXML.getCharacterStream	Reader	无
SQLXML.getSource	Source[1]	无
SQLXML.getString	String	无
DB2Xml.getDB2AsciiStream	InputStream	无
DB2Xml.getDB2BinaryStream	InputStream	无
DB2Xml.getDB2Bytes	byte[]	无
DB2Xml.getDB2CharacterStream	Reader	无
DB2Xml.getDB2String	String	无
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	由 getDB2XmlBinaryStream *targetEncoding* 参数指定
DB2Xml.getDB2XmlBytes	byte[]	由 DB2Xml.getDB2XmlBytes *targetEncoding* 参数指定
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	String	ISO-10646-UCS-2

注:

1. 所返回的类由 getSource 的调用程序指定，但是该类必须扩展 javax.xml.transform.Source。

如果应用程序对要返回的数据执行 XMLSERIALIZE 函数，那么在执行该函数之后，这些数据将具有 XMLSERIALIZE 函数中指定的数据类型，而不是 XML 数据类型。因此，驱动程序将把数据作为指定类型来处理，并且会忽略任何内部编码声明。

示例: 以下示例说明将 XML 列中的数据检索到 SQLXML 对象中，然后使用 SQLXML.getString 方法将数据检索到字符串中。

```
public void fetchToSQLXML(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToSQLXML: Get XML data as an SQLXML object " +
    "using getSQLXML");
        PreparedStatement selectStmt = null;
        String sqls = null, stringDoc = null;
        ResultSet rs = null;

        try{
            sqls = "SELECT info FROM customer WHERE cid = " + cid;
            selectStmt = conn.prepareStatement(sqls);
            rs = selectStmt.executeQuery();
```

```
      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
            ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToSQLXML: Column type = " + colType);
            while (rs.next()) {
        // Retrieve the XML data with getSQLXML.
        // Then write it to a string with
              // explicit internal ISO-10646-UCS-2 encoding.
        java.sql.SQLXML xml = rs.getSQLXML(1);
        System.out.println (xml.getString());
      }
            rs.close();
    }
        catch (SQLException sqle) {
      System.out.println("fetchToSQLXML: SQL Exception: " +
                sqle.getMessage());
      System.out.println("fetchToSQLXML: SQL State: " +
                sqle.getSQLState());
      System.out.println("fetchToSQLXML: SQL Error Code: " +
                sqle.getErrorCode());
    }
  }
```

示例: 以下示例说明将 XML 列中的数据检索到 String 变量中。

```
public void fetchToString(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToString: Get XML data " +
        "using getString");
        PreparedStatement selectStmt = null;
        String sqls = null, stringDoc = null;
        ResultSet rs = null;

        try{
            sqls = "SELECT info FROM customer WHERE cid = " + cid;
            selectStmt = conn.prepareStatement(sqls);
            rs = selectStmt.executeQuery();

      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
            ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
            System.out.println("fetchToString: Column type = " + colType);

            while (rs.next()) {
                stringDoc = rs.getString(1);
                System.out.println("Document contents:");
                System.out.println(stringDoc);
 }
        catch (SQLException sqle) {
            System.out.println("fetchToString: SQL Exception: " +
                sqle.getMessage());
            System.out.println("fetchToString: SQL State: " +
                sqle.getSQLState());
            System.out.println("fetchToString: SQL Error Code: " +
                sqle.getErrorCode());
    }
  }
```

示例: 以下示例说明将 XML 列中的数据检索到 DB2Xml 对象中，然后使用 DB2Xml.getDB2XmlString 方法来将数据检索到 String 对象中，并且添加了符合 ISO-10646-UCS-2 编码规范的 XML 声明。

```
public void fetchToDB2Xml(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToDB2Xml: Get XML data as a DB2XML object " +
        "using getObject");
        PreparedStatement selectStmt = null;
        String sqls = null, stringDoc = null;
        ResultSet rs = null;

        try{
            sqls = "SELECT info FROM customer WHERE cid = " + cid;
            selectStmt = conn.prepareStatement(sqls);
            rs = selectStmt.executeQuery();

      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
            ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToDB2Xml: Column type = " + colType);
            while (rs.next()) {
              // Retrieve the XML data with getObject, and cast the object
              // as a DB2Xml object. Then write it to a string with
      // explicit internal ISO-10646-UCS-2 encoding.
                com.ibm.db2.jcc.DB2Xml xml =
                    (com.ibm.db2.jcc.DB2Xml) rs.getObject(1);
                System.out.println (xml.getDB2XmlString());
      }
            rs.close();
    }
        catch (SQLException sqle) {
      System.out.println("fetchToDB2Xml: SQL Exception: " +
                sqle.getMessage());
      System.out.println("fetchToDB2Xml: SQL State: " +
                sqle.getSQLState());
      System.out.println("fetchToDB2Xml: SQL Error Code: " +
                sqle.getErrorCode());
    }
  }
```

在 Java 应用程序中调用具有 XML 参数的例程

Java 应用程序可以调用具有 XML 参数的 DB2 数据库 Linux 版、UNIX 版和 Windows 版数据源中的存储过程。

对于本机 SQL 过程，存储过程定义中的 XML 参数为 XML 类型。而对于 DB2 数据库 Linux 版、UNIX 版和 Windows 版数据源上的外部存储过程和用户定义的函数，例程定义中的 XML 参数为 XML AS CLOB 类型。当调用具有 XML 参数的存储过程或用户定义的函数时，需要在调用语句中使用兼容数据类型。

要从 JDBC 程序中调用具有 XML 输入参数的例程，请使用 java.sql.SQLXML 或 com.ibm.db2.jcc.DB2Xml 类型的参数。要注册 XML 输出参数，请将这些参数注册为 java.sql.Types.SQLXML 或 com.ibm.db2.jcc.DB2Types.XML 类型。（不推荐使用 com.ibm.db2.jcc.DB2Xml 和 com.ibm.db2.jcc.DB2Types.XML 类型。）

示例：将调用具有三个 XML 参数（IN、OUT 和 INOUT 参数）的存储过程的 JDBC 程序。此示例需要 JDBC 4.0。

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                  // Declare an input, output, and
                                  // INOUT XML parameter
Connection con;
CallableStatement cstmt;
```

```
ResultSet rs;
...
cstmt = con.prepareCall("CALL SP_xml(?,?,?)");
                                  // Create a CallableStatement object
cstmt.setObject (1, in_xml);      // Set input parameter
cstmt.setObject (3, inout_xml);   // Set inout parameter
cstmt.registerOutParameter (2, java.sql.Types.SQLXML);
                                  // Register out and input parameters
cstmt.registerOutParameter (3, java.sql.Types.SQLXML);
cstmt.executeUpdate();            // Call the stored procedure
out_xml = cstmt.getSQLXML(2);     // Get the OUT parameter value
inout_xml = cstmt.getSQLXML(3);   // Get the INOUT parameter value
System.out.println("Parameter values from SP_xml call: ");
System.out.println("Output parameter value ");
MyUtilities.printString(out_xml.getString());
                          // Use the SQLXML.getString
                          // method to convert the out_xml
                          // value to a string for printing.
                          // Call a user-defined method called
                          // printString (not shown) to print
                          // the value.
System.out.println("INOUT parameter value ");
MyUtilities.printString(inout_xml.getString());
                          // Use the SQLXML.getString
                          // method to convert the inout_xml
                          // value to a string for printing.
                          // Call a user-defined method called
                          // printString (not shown) to print
                          // the value.
```

要从 SQLJ 程序中调用具有 XML 参数的例程，请使用 java.sql.SQLXML 或 com.ibm.db2.jcc.DB2Xml 类型的参数。

示例: 将调用具有三个 XML 参数（IN、OUT 和 INOUT 参数）的存储过程的 SQLJ 程序。此示例需要 JDBC 4.0。

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                  // Declare an input, output, and
                                  // INOUT XML parameter
...
#sql [myConnCtx] {CALL SP_xml(:IN in_xml,
                            :OUT out_xml,
                            :INOUT inout_xml)};
                                  // Call the stored procedure
System.out.println("Parameter values from SP_xml call: ");
System.out.println("Output parameter value ");
MyUtilities.printString(out_xml.getString());
                          // Use the SQLXML.getString
                          // method toconvert the out_xml value
                          // to a string for printing.
                          // Call a user-defined method called
                          // printString (not shown) to print
                          // the value.
System.out.println("INOUT parameter value ");
MyUtilities.printString(inout_xml.getString());
                          // Use the SQLXML.getString
                          // method to convert the inout_xml
                          // value to a string for printing.
                          // Call a user-defined method called
                          // printString (not shown) to print
                          // the value.
```

SQLJ

SQLJ 应用程序中的 XML 数据

在 SQLJ 应用程序中，可以将数据存储在 XML 列中并从 XML 列中检索数据。

在 DB2 表中，XML 内置数据类型用于将 XML 数据以一组采用树形结构的结构化节点形式存储在列中。

在应用程序中，XML 数据采用已序列化字符串格式。

在连接至 DB2 数据库 Linux 版、UNIX 版和 Windows 版的 SQLJ 应用程序中，XML 数据采用文本 XML 格式。在连接至 DB2 z/OS 版的 SQLJ 应用程序中，XML 数据可以采用文本 XML 格式或者二进制 XML 格式。

在 SQLJ 应用程序中，您可以执行下列操作:

- 使用 INSERT、UPDATE 或 MERGE 语句将整个 XML 文档存储在 XML 列中。
- 使用单行 SELECT 语句或迭代器从 XML 列中检索整个 XML 文档。
- 通过以下方法从 XML 列中的文档内检索一个序列: 使用 SQL XMLQUERY 函数在数据库中检索该序列，然后使用单行 SELECT 语句或迭代器将已序列化的 XML 字符串数据检索到应用程序变量中。
- 通过以下方法从 XML 列中的文档内检索一个序列: 使用前面添加了字符串"XQUERY"的 XQuery 表达式来将该序列的元素检索到数据库中的结果表中，而结果表中的每一行都表示该序列中的一项。然后使用单行 SELECT 语句或迭代器来将数据检索到应用程序变量中。
- 通过以下方法从 XML 列中的文档内检索一个作为用户定义的表的序列: 使用 SQL XMLTABLE 函数来定义并检索结果表。然后使用单行 SELECT 语句或迭代器来将结果表中的数据检索到应用程序变量中。

可以使用 JDBC 4.0 `java.sql.SQLXML` 对象来检索和更新 XML 列中的数据。如果调用诸如 ResultSetMetaData.getColumnTypeName 之类的元数据方法，那么会对 XML 列类型返回整数值 `java.sql.Types.SQLXML`。

SQLJ 应用程序中的 XML 列更新

当您更新数据库表的 XML 列或者向这些列中插入数据时，SQLJ 应用程序中的输入数据必须采用已序列化的字符串格式。

可以用来更新 XML 列的主机表达式数据类型包括:

- java.sql.SQLXML（需要 SDK for Java V6 或更新版本，以及 IBM 数据服务器 JDBC 和 SQLJ 驱动程序版本 4.0 或更新版本）
- com.ibm.db2.jcc.DB2Xml（不推荐使用）
- String
- byte
- Blob
- Clob
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

XML 数据的编码可以从数据本身派生（称为*内部编码*数据），也可以从外部源派生（称为*外部编码*数据）。作为二进制数据发送至数据库服务器的 XML 数据被当作内部编码的数据来处理。作为字符数据发送至数据源的 XML 数据被当作外部编码的数据来处理。JVM 的缺省编码是外部编码。

Java 应用程序的外部编码始终为 Unicode 编码。

外部编码的数据可以具有内部编码。即，可以将数据作为字符数据发送至数据源，但该数据中包含编码信息。数据源按如下所示来处理内部编码与外部编码之间的不兼容性:

- 如果数据源是 DB2 数据库 Linux 版、UNIX 版和 Windows 版，那么在外部编码与内部编码不兼容的情况下，数据源将产生错误，除非外部编码和内部编码都是 Unicode。如果外部编码和内部编码都是 Unicode，那么数据源将忽略内部编码。
- 如果数据源是 DB2 z/OS 版，那么它将忽略内部编码。

XML 列中的数据使用 UTF-8 编码来存储。

示例: 假定您使用以下语句将 String 主机表达式 xmlString 中的数据插入到一个表的 XML 列中。xmlString 是字符类型，因此，无论它是否具有内部编码规范，都将使用它的外部编码。

```
#sql [ctx] {INSERT INTO CUSTACC VALUES (1, :xmlString)};
```

示例: 假定您将数据从 xmlString 复制到使用 CP500 编码的字节数组中。数据包含 XML 声明和 CP500 的编码声明。然后，将 byte[] 主机表达式中的数据插入到一个表的 XML 列中。

```
byte[] xmlBytes = xmlString.getBytes("CP500");
#sql[ctx] {INSERT INTO CUSTACC VALUES (4, :xmlBytes)};
```

字节字符串被认为是在内部编码的数据。如果需要，数据将从它的内部编码方案转换为 UTF-8，并采用分层格式存储在数据源上。

示例: 假定您将数据从 xmlString 复制到使用 US-ASCII 编码的字节数组中。然后，构造一个 sqlj.runtime.AsciiStream 主机表达式，并将 sqlj.runtime.AsciiStream 主机表达式中的数据插入到数据源上的一个表的 XML 列中。

```
byte[] b = xmlString.getBytes("US-ASCII");
java.io.ByteArrayInputStream xmlAsciiInputStream =
    new java.io.ByteArrayInputStream(b);
sqlj.runtime.AsciiStream sqljXmlAsciiStream =
    new sqlj.runtime.AsciiStream(xmlAsciiInputStream,  b.length);
#sql[ctx] {INSERT INTO CUSTACC VALUES (4, :sqljXmlAsciiStream)};
```

sqljXmlAsciiStream 是一种流类型，因此将使用它的内部编码。数据将从它的内部编码转换为 UTF-8 编码，并采用分层格式存储在数据源上。

示例: sqlj.runtime.CharacterStream 主机表达式: 假定您构造一个 sqlj.runtime.CharacterStream 主机表达式，并将 sqlj.runtime.CharacterStream 主机表达式中的数据插入到一个表的 XML 列中。

```
java.io.StringReader xmlReader =
    new java.io.StringReader(xmlString);
sqlj.runtime.CharacterStream sqljXmlCharacterStream =
    new sqlj.runtime.CharacterStream(xmlReader,  xmlString.length());
#sql [ctx] {INSERT INTO CUSTACC VALUES (4, :sqljXmlCharacterStream)};
```

sqljXmlCharacterStream 是字符类型，因此，无论它是否具有内部编码规范，都将使用它的外部编码。

示例：假定您将一个文档从 XML 列检索到 java.sql.SQLXML 主机表达式中，并将数据插入到一个表的 XML 列中。

```
java.sql.ResultSet rs = s.executeQuery ("SELECT * FROM CUSTACC");
rs.next();
java.sql.SQLXML xmlObject = (java.sql.SQLXML)rs.getObject(2);
#sql [ctx] {INSERT INTO CUSTACC VALUES (6, :xmlObject)};
```

检索数据之后，它仍然采用 UTF-8 编码。因此，当您将该数据插入另一个 XML 列时，不会进行转换。

示例：假定您将一个文档从 XML 列检索到 com.ibm.db2.jcc.DB2Xml 主机表达式中，并将数据插入到一个表的 XML 列中。

```
java.sql.ResultSet rs = s.executeQuery ("SELECT * FROM CUSTACC");
rs.next();
com.ibm.db2.jcc.DB2Xml xmlObject = (com.ibm.db2.jcc.DB2Xml)rs.getObject(2);
#sql [ctx] {INSERT INTO CUSTACC VALUES (6, :xmlObject)};
```

检索数据之后，它仍然采用 UTF-8 编码。因此，当您将该数据插入另一个 XML 列时，不会进行转换。

在 JDBC 应用程序中检索 XML 数据

在 SQLJ 应用程序中，从数据库表的 XML 列中检索数据时，输出数据必须已显式或隐式序列化。

可以用来从 XML 列中检索数据的主机表达式或迭代器数据类型包括:
- java.sql.SQLXML（需要 SDK for Java V6 或更新版本，以及 IBM 数据服务器 JDBC 和 SQLJ 驱动程序版本 4.0 或更新版本）
- com.ibm.db2.jcc.DB2Xml（不推荐使用）
- String
- byte[]
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

如果在检索数据之前应用程序不调用 XMLSERIALIZE 函数，那么数据将从 UTF-8 转换为字符数据类型的外部应用程序编码，或者转换为二进制数据类型的内部编码。不会添加 XML 声明。如果主机表达式是一个 java.sql.SQLXML 或 com.ibm.db2.jcc.DB2Xml 类型的对象，那么需要调用其他方法来从此对象中检索数据。您调用的方法将确定输出数据的编码以及是否添加了具有编码规范的 XML 声明。

下表列示了一些方法，可以通过调用这些方法来从 java.sql.SQLXML 或 com.ibm.db2.jcc.DB2Xml 对象中检索数据，以及检索相应的输出数据类型和 XML 声明中的编码类型。

表 35. SQLXML 和 DB2Xml 方法、数据类型以及添加的编码规范

方法	输出数据类型	已添加的 XML 内部编码声明的类型
SQLXML.getBinaryStream	InputStream	无
SQLXML.getCharacterStream	Reader	无

表 35. SQLXML 和 DB2Xml 方法、数据类型以及添加的编码规范 (续)

方法	输出数据类型	已添加的 XML 内部编码声明的类型
SQLXML.getSource	Source	无
SQLXML.getString	String	无
DB2Xml.getDB2AsciiStream	InputStream	无
DB2Xml.getDB2BinaryStream	InputStream	无
DB2Xml.getDB2Bytes	byte[]	无
DB2Xml.getDB2CharacterStream	Reader	无
DB2Xml.getDB2String	String	无
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	由 getDB2XmlBinaryStream *targetEncoding* 参数指定
DB2Xml.getDB2XmlBytes	byte[]	由 DB2Xml.getDB2XmlBytes *targetEncoding* 参数指定
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	String	ISO-10646-UCS-2

如果应用程序对要返回的数据执行 XMLSERIALIZE 函数，那么在执行该函数之后，这些数据将具有 XMLSERIALIZE 函数中指定的数据类型，而不是 XML 数据类型。因此，驱动程序将把数据作为指定类型来处理，并且会忽略任何内部编码声明。

示例: 假定您将 XML 列中的数据检索到 String 主机表达式中。

```
#sql iterator  XmlStringIter (int, String);
#sql [ctx] siter = {SELECT C1, CADOC from CUSTACC};
#sql {FETCH :siter INTO :row, :outString};
```

String 类型是字符类型，因此，数据将从 UTF-8 转换为外部编码（这是缺省 JVM 编码），并且返回时不带任何 XML 声明。

示例: 假定您将 XML 列中的数据检索到 byte[] 主机表达式中。

```
#sql iterator  XmlByteArrayIter (int, byte[]);
XmlByteArrayIter biter = null;
#sql [ctx] biter =  {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :biter INTO :row, :outBytes};
```

byte[] 类型是二进制类型，因此不会转换使用 UTF-8 编码的数据，并且数据返回时不带任何 XML 声明。

示例: 假定您将一个文档从 XML 列检索到 java.sql.SQLXML 主机表达式中，但您需要二进制流中的数据。

```
#sql iterator  SqlXmlIter (int, java.sql.SQLXML);
SqlXmlIter SQLXMLiter = null;
java.sql.SQLXML outSqlXml = null;
#sql [ctx] SqlXmlIter = {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :SqlXmlIter INTO :row, :outSqlXml};
java.io.InputStream XmlStream = outSqlXml.getBinaryStream();
```

FETCH 语句将数据检索到采用 UTF-8 编码的 SQLXML 对象中。SQLXML.getBinaryStream 将数据存储在二进制流中。

示例：假定您将一个文档从 XML 列检索到 com.ibm.db2.jcc.DB2Xml 主机表达式中，但是您需要具有 XML 声明的字节字符串中的数据，而 XML 声明包含 UTF-8 的内部编码规范。

```
#sql iterator  DB2XmlIter (int, com.ibm.db2.jcc.DB2Xml);
DB2XmlIter db2xmliter = null;
com.ibm.db2.jcc.DB2Xml outDB2Xml = null;
#sql [ctx] db2xmliter = {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :db2xmliter INTO :row, :outDB2Xml};
byte[] byteArray = outDB2XML.getDB2XmlBytes("UTF-8");
```

FETCH 语句将数据检索到采用 UTF-8 编码的 DB2Xml 对象中。具有 UTF-8 参数的 getDB2XmlBytes 方法将添加具有 UTF-8 编码规范的 XML 声明，并将数据存储在字节数组中。

PHP

IBM 数据服务器的 PHP 应用程序开发

PHP：超文本预处理器（PHP）是一种解释型编程语言，广泛用于开发 Web 应用程序。PHP 已经成为用于 Web 开发的一种流行语言，因为它易于学习，主要提供实用的解决方案，并且支持 Web 应用程序中需要的最常见功能。

PHP 是一种模块化语言，使您能够通过使用扩展来定义可用功能。这些扩展可以简化诸如下列任务：读写和处理 XML，创建 SOAP 客户机和服务器，以及对服务器和浏览器之间的通信进行加密。然而，PHP 的最流行的扩展提供了对数据库的读写访问权，因此，您可以很轻松地创建动态数据库驱动的 Web 站点。

IBM 提供了下列 PHP 扩展来访问 IBM 数据服务器数据库：

ibm_db2
: 一个过程 应用程序编程接口（API），除了执行正常的创建、读取、更新和写入数据库操作以外，还提供了对数据库元数据的大量访问。可以使用 PHP 4 或 PHP 5 来编译 ibm_db2 扩展。此扩展是由 IBM 编写、维护和支持的一个扩展。

pdo_ibm
: 一个用于 PHP 数据对象（PDO）扩展的驱动程序，它通过 PHP 5.1 中引入的面向对象的标准数据库接口来访问 IBM 数据服务器数据库。

这些扩展都是 IBM 数据服务器客户机的一部分。还可以从 PHP 扩展公用库（PECL）http://pecl.php.net/ 中获得最新版本的 ibm_db2 和 pdo_ibm。

PHP 应用程序可以访问下列 IBM 数据服务器数据库：

- IBM DB2 版本 9.1 Linux 版、UNIX 版和 Windows 版修订包 Pack 2 和更高版本
- IBM DB2® Universal Database™（DB2 UDB）版本 8 Linux 版、UNIX 版和 Windows 版修订包 15 和更高版本
- 与 i5/OS® V5R3 上的 IBM DB2 Universal Database的远程连接
- 与 IBM DB2 for IBM i 5.4 和更高版本的远程连接
- 与 IBM DB2 z/OS 版版本 8 和版本 9 的远程连接

第三个扩展"统一 ODBC"原先就提供了对 DB2 数据库系统的访问权。但是，对于新的应用程序，应使用 ibm_db2 和 pdo_ibm，这是因为它们在性能和稳定性方面都比"统一 ODBC"具有更大的优势。通过 ibm_db2 扩展 API 来移植先前为"统一 ODBC"编写的应用程序，几乎就像在应用程序的所有源代码中将 `odbc_` 函数名全局更改为 `db2_` 一样容易。

使用 PHP 应用程序检索 XML 数据

ibm_db2 API 是用于访问 IBM 数据服务器数据库的 PHP 扩展。ibm_db2 API 支持连接至 DB2 数据库并且支持从 XML 列返回 XML 数据。

ibm_db2 API 还支持使用诸如 XMLTABLE 的 DB2 函数以及支持在 SQL 语句中执行 XQuery 表达式。在 SQL 语句中，使用 XMLQUERY 函数调用 XQuery 表达式。

有关使用 ibm_db2 API 开发 PHP 应用程序的更多信息，请参阅"使用 ibm_db2 进行 PHP 应用程序开发"*Developing Perl, PHP, Python, and Ruby on Rails Applications*。

PHP 下载和相关资源

提供了许多资源以帮助您开发用于 IBM 数据服务器的 PHP 应用程序。

表 36. PHP 下载和相关资源

下载	
完整 PHP 源代码(包括 Windows 二进制。已经对随 PHP 一起提供的大多数 Linux® 分发对象进行了预编译。)	http://www.php.net/downloads.php
PHP 扩展公用库（PECL）中的 ibm_db2 和 pdo_ibm	http://pecl.php.net/
IBM 数据服务器客户机	http://www-01.ibm.com/software/data/db2/ad/deploy.html
修订包和客户机下载	http://www-01.ibm.com/support/docview.wss?rs=71&uid=swg27007053
Zend Server	http://www.zend.com/en/products/server/downloads
文档	
PHP 手册	http://www.php.net/docs.php
ibm_db2 API 文档	http://www.php.net/ibm_db2
PDO API 文档	http://php.net/manual/en/book.pdo.php
相关资源	
PHP Web 站点	http://www.php.net/

Perl

pureXML 和 Perl

DBD::DB2 驱动程序支持 DB2 pureXML。对于 pureXML 的支持允许通过 DBD::DB2 驱动程序更直接地访问数据，并通过在应用程序与数据库之间提供更多透明通信来帮助减少应用程序逻辑。

借助 pureXML 支持，可直接将 XML 文档插入到 DB2 数据库中。因为 pureXML 解析器会在您将 XML 数据插入到数据库中时自动运行，所以应用程序不再需要解析 XML 文档。将文档解析放在应用程序外部进行可改进应用程序性能并降低维护工作量。使用 DBD::DB2 驱动程序检索 XML 存储数据也很简单；可使用 BLOB 或记录来访问数据。

有关 DB2 Perl 数据库接口以及如何下载最新 DBD::DB2 驱动程序的信息，请参阅 http://www.ibm.com/software/data/db2/perl。

示例

以下示例是一个使用 pureXML 的 Perl 程序:

```
#!/usr/bin/perl
use DBI;
use strict ;

# Use DBD:DB2 module:
#   to create a simple DB2 table with an XML column
#   Add one row of data
#   retreive the XML data as a record or a LOB (based on $datatype).

# NOTE: the DB2 SAMPLE database must already exist.

my $database='dbi:DB2:sample';
my $user='';
my $password='';

my $datatype = "record" ;
#  $datatype = "LOB" ;

my $dbh = DBI->connect($database, $user, $password)
  or die "Can't connect to $database: $DBI::errstr";

# For LOB datatype, LongReadLen = 0 -- no data is retrieved on initial fetch
$dbh->{LongReadLen} = 0 if $datatype eq "LOB" ;

# SQL CREATE TABLE to create test table
my $stmt = "CREATE TABLE xmlTest (id INTEGER, data XML)";
my $sth = $dbh->prepare($stmt);
$sth->execute();

#insert one row of data into table
insertData() ;

# SQL SELECT statement returns home phone element from XML data
$stmt = qq(
 SELECT XMLQUERY ('
 \$d/*:customerinfo/*:phone[\@type = "home"] '
 passing data as "d")
 FROM xmlTest
) ;


# prepare and execute SELECT statement
$sth = $dbh->prepare($stmt);
$sth->execute();

# Print data returned from select statement
if($datatype eq "LOB") {
    printLOB() ;
}
else {
 printRecord() ;
```

```
}

# Drop table
$stmt = "DROP TABLE xmlTest" ;
$sth = $dbh->prepare($stmt);
$sth->execute();

warn $DBI::errstr if $DBI::err;

$sth->finish;
$dbh->disconnect;


##############


sub printRecord {
 print "output data as as record\n" ;

 while( my @row = $sth->fetchrow )
 {
  print $row[0] . "\n";
 }

 warn $DBI::errstr if $DBI::err;
}


sub printLOB {
 print "output as Blob data\n" ;

 my $offset = 0;
 my $buff="";
 $sth->fetch();
 while( $buff = $sth->blob_read(1,$offset,1000000)) {
  print $buff;
  $offset+=length($buff);
  $buff="";
 }
 warn $DBI::errstr if $DBI::err;
}


sub insertData {

 # insert a row of data
 my $xmlInfo = qq(\'
 <customerinfo xmlns="http://posample.org" Cid="1011">
   <name>Bill Jones</name>
<addr country="Canada">
     <street>5 Redwood</street>
<city>Toronto</city>
<prov-state>Ontario</prov-state>
     <pcode-zip>M6W 1E9</pcode-zip>
   </addr>
   <phone type="work">416-555-9911</phone>
   <phone type="home">416-555-1212</phone>
 </customerinfo>
 \') ;

 my $catID = 1011 ;

 # SQL statement to insert data.
 my $Sql = qq(
  INSERT INTO xmlTest (id, data)
      VALUES($catID, $xmlInfo )
 );
```

```
 $sth = $dbh->prepare( $Sql )
   or die "Can't prepare statement: $DBI::errstr";

 my $rc = $sth->execute
   or die "Can't execute statement: $DBI::errstr";

 # check for problems
 warn $DBI::errstr if $DBI::err;
}
```

Perl 中的数据库连接

DBD::DB2 驱动程序支持由 DBI API 定义的标准数据库连接函数。

要使 Perl 能够装入 DBI 模块，必须在应用程序中包含以下代码行:

```
use DBI;
```

当按照以下语法使用 DBI->connect 语句来创建数据库句柄时，DBI 模块将自动装入 DBD::DB2 驱动程序:

```
my $dbhandle = DBI->connect('dbi:DB2:dsn', $userID, $password);
```

其中:

$dbhandle
: 表示 connect 语句所返回的数据库句柄

dsn
: 对于本地连接，dsn 表示在 DB2 数据库目录中编目的 DB2 别名

  对于远程连接，dsn 表示一个完整的连接字符串，其中包括用于连接至远程主机的主机名、端口号、协议、用户标识和密码

$userID
: 表示用于连接至数据库的用户标识

$password
: 表示用于连接至数据库的用户标识的密码

有关 DBI API 的更多信息，请参阅http://search.cpan.org/~timb/DBI/DBI.pm。

示例

示例 1: 连接至本地主机上的数据库（客户机和服务器位于同一工作站上）

```
use DBI;

$DATABASE = 'dbname';
$USERID = 'username';
$PASSWORD = 'password';

my $dbh = DBI->connect("dbi:DB2:$DATABASE", $USERID, $PASSWORD, {PrintError => 0})
or die "Couldn't connect to database: " . DBI->errstr;

$dbh->disconnect;
```

示例 2: 连接至远程主机上的数据库（客户机和服务器位于不同工作站上）

```
use DBI;

$DSN="DATABASE=sample; HOSTNAME=host; PORT=60000; PROTOCOL=TCPIP; UID=username;
```

```
PWD=password";

my $dbh = DBI->connect("dbi:DB2:$DSN", $USERID, $PASSWORD, {PrintError => 0})
or die "Couldn't connect to database: " . DBI->errstr;

$dbh->disconnect;
```

Perl 限制

某些限制适用于使用 Perl 开发应用程序时可用的支持。

Perl DBI 模块仅支持动态 SQL。当您必须多次执行某个语句时，可通过发出 prepare 调用来对该语句进行预编译，从而改进 Perl 应用程序的性能。

Perl 不支持多线程数据库访问。

有关工作站上所安装 DBD::DB2 驱动程序版本的限制的更多信息，请参阅 DBD::DB2 驱动程序包中的 CAVEATS 文件。

例程

SQL 过程

SQL 过程中的 XML 和 XQuery 支持

SQL 过程支持使用 XML 数据类型的参数和变量。在 SQL 语句中可以像使用任何其他数据类型的变量一样使用它们。另外，可以将 XML 数据类型的变量作为参数传递给 XMLEXISTS、XMLQUERY 和 XMLTABLE 表达式中的 XQuery 表达式。

以下示例说明了 SQL 过程中的 XML 参数和变量的声明、使用和赋值:

```
  CREATE TABLE T1(C1 XML) %

  CREATE PROCEDURE proc1(IN parm1 XML, IN parm2 VARCHAR(32000))
LANGUAGE SQL
BEGIN
        DECLARE var1 XML;

        /* check if the value of XML parameter parm1
              contains an item with a value less than 200 */
   IF(XMLEXISTS('$x/ITEM[value < 200]' passing by ref parm1 as "x"))THEN

     /* if it does, insert the value of parm1 into table T1 */
     INSERT INTO T1 VALUES(parm1);

   END IF;

   /* parse parameter parm2's value and assign it to a variable */
   SET var1 = XMLPARSE(document parm2 preserve whitespace);

   /* insert variable var1 into table T1
   INSERT INTO T1 VALUES(var1);

END %
```

在上述示例中，有一个具有 XML 列的表 T1。SQL 过程接受数据类型为 XML 的两个参数：parm1 和 parm2。在 SQL 过程中，一个 XML 变量被声明为 var1。

SQL 过程的逻辑将检查 XML 参数 parm1 的值是否包含一个值小于 200 的项。如果包含这样一项，那么会将该 XML 值直接插入到 T1 表的 C1 列中。

然后，将使用 XMLPARSE 函数来解析 parm2 参数的值，并将该值作为 XML 变量 var1 的赋值。然后，还会将该 XML 变量值插入到 T1 表的 C1 列中。

对 XQuery 操作实现控制流逻辑这项能力，使得很容易开发用来查询和访问存储在数据库中的 XML 数据的复杂算法。

SQL 过程中 XQuery 表达式的游标

SQL 过程支持在 XQuery 表达式上定义游标。XQuery 表达式上的游标允许您对该表达式返回的 XQuery 序列的元素进行迭代。

可以静态或动态定义 SQL 语句上的游标，但只能动态定义 XQuery 表达式上的游标。要动态声明一个游标，需要声明一个 CHAR 或 VARCHAR 类型的变量，以包含将用来定义游标结果集的 XQuery 表达式。必须准备好 XQuery 表达式之后才能打开游标和解析结果集。

以下是一个 SQL 过程示例，它将为 XQuery 表达式动态声明一个游标、打开该游标并访存 XML 数据:

```
CREATE PROCEDURE xmlProc(IN inCust XML, OUT resXML XML)
SPECIFIC xmlProc
LANGUAGE SQL
BEGIN
        DECLARE SQLSTATE CHAR(5);
      DECLARE stmt_text VARCHAR (1024);
        DECLARE customer XML;
        DECLARE cityXml XML;
        DECLARE city VARCHAR (100);
      DECLARE stmt STATEMENT;
      DECLARE cur1 CURSOR FOR stmt;

      -- Get the city of the input customer
   SET cityXml = XMLQUERY('$cust/customerinfo//city' passing inCust as "cust");
      SET city = XMLCAST(cityXml as VARCHAR(100));

    -- Iterate over all the customers from the city using an XQUERY cursor
    -- and collect the customer name values into the output XML value

   SET stmt_text = 'XQUERY for $cust
                                     in db2-fn:xmlcolumn("CUSTOMER.INFO")
                      /*:customerinfo/*:addr[*:city= "' || city ||'"]
                   return <Customer>{$cust/../@Cid}{$cust/../*:name}</Customer>';

    -- Use the name of the city for the input customer data as a prefix
    SET resXML = cityXml;

    PREPARE stmt FROM stmt_text;
    OPEN cur1;

      FETCH cur1 INTO customer;
    WHILE (SQLSTATE = '00000') DO
        SET resXML = XMLCONCAT(resXML, customer);
        FETCH cur1 INTO customer;
    END WHILE;

  set resXML = XMLQUERY('<result> {$res} </result>'
                                                passing resXML as "res");

  END
```

此 SQL 过程将收集在表名 CUSTOMER 中定义的客户的标识和姓名，这些客户与将 XML 数据作为输入参数来提供的那些客户处于同一城市。

可以按如下所示执行 CALL 语句来调用上述 SQL 过程：

```
CALL xmlProc(xmlparse(document '<customerinfo Cid="5002">
                                <name>Jim Noodle</name>
                                <addr country="Canada">
                                   <street>25 EastCreek</street>
                                   <city>Markham</city>
                                   <prov-state>Ontario</prov-state>
                                   <pcode-zip>N9C-3T6</pcode-zip>
                                </addr>
                                <phone type="work">905-566-7258</phone>
                              </customerinfo>' PRESERVE WHITESPACE),?)
```

如果创建了此 SQL 过程并且对 SAMPLE 数据库运行了它，就会返回两个客户的 XML 数据。

由于 XML 值不支持参数标记，此局限性的一个变通方法是在包括一个或多个局部变量值的已并置语句片段外部构造一个动态 SQL 语句。

例如：

```
DECLARE person_name VARCHAR(128);

SET person_name = "Joe";
SET stmt_text = 'XQUERY for $fname in db2-fn:sqlquery
       ("SELECT doc
              FROM T1
      WHERE DOCID=1")//fullname where $fname/first = ''' person_name || ''';
```

此示例在包含 SQL 全查询的 XQuery 语句的变量赋值中返回一个结果集。该结果集包含名字为 Joe 的人员的全名。从功能上来说，SQL 部分将从 T1 表的 doc 列中选择标识为 1 的 XML 文档。然后，XQuery 部分在 XML 文档中选择 first 值为 Joe 的 fullname 值。

落实和回滚对 SQL 过程中的 XML 参数和变量值的作用

SQL 过程中的落实和回滚会影响数据类型为 XML 的参数和变量的值。在执行 SQL 过程期间，一旦执行落实或回滚操作，为 XML 参数和 XML 变量指定的值就不再可用。

在执行落实或回滚操作之后，如果尝试引用 XML 数据类型的 SQL 变量或 SQL 参数，那么将发生错误（SQL1354N，560CE）。

要在执行落实或回滚操作之后成功地引用 XML 参数和变量，必须首先为它们指定新值。

当将 ROLLBACK 和 COMMIT 语句添加至 SQL 过程时，应考虑 XML 参数和变量值的可用性。

SQL 函数

SQL 函数中数据类型为 XML 的参数和变量

DB2 支持您使用 CREATE FUNCTION（SQL 标量、表或行）语句或者 CREATE FUNCTION（有源或模板）语句创建的内联型 SQL 函数的 XML 数据类型。

在使用 CREATE FUNCTION（SQL 标量、表或行）语句创建的内联型用户定义的函数中，可以在 SQL 语句中像使用任何其他数据类型的变量一样使用 XML 变量。例如，在用户定义的函数中，可以将数据类型为 XML 的变量作为参数传递给 XMLEXISTS 谓词或者诸如 XMLQUERY 或 XMLTABLE 等函数中的 XQuery 表达式。

在使用 CREATE FUNCTION（有源或模板）语句创建的用户定义的函数中（其中，源函数是用户定义的 SQL 标量函数），可以将 XML 数据类型用作输入、输出或者输入/输出参数。

XML 值由用户定义的函数中的引用指定。

数据类型为 XML 的参数和变量在编译型 SQL 函数中不受支持。

示例

以下示例函数是一个内联 SQL 标量函数，它使用 XML 数据类型作为输入参数和变量。该函数将使用 XQuery 表达式从 XML 文档中抽取 phone number 元素并返回该 phone number 元素:

```
CREATE FUNCTION phone_number ( dept_doc XML )
RETURNS XML
LANGUAGE SQL
NO EXTERNAL ACTION
BEGIN ATOMIC
DECLARE tmp_xml XML;
 IF (XMLEXISTS('$test/department/phone' passing by ref dept_doc as "test"))
 THEN
  SET tmp_xml = XMLQUERY('document
       {<phone_list>{$doc/department/phone}</phone_list>}'
    PASSING dept_doc as "doc");
 ELSE
 SET tmp_xml = XMLPARSE(document '<phone_list><phone>N/A</phone></phone_list>');
END IF;
RETURN tmp_xml;
END
```

以下 SELECT 语句将使用 PHONE_NUMBER 函数从具有职员信息的表中的 XML 文档中检索电话号码。

```
SELECT PHONE_NUMBER(info) FROM employees WHERE empid = 12356
```

此 SELECT 语句假定此表与使用以下 CREATE TABLE 语句创建的表相似，并且此表包含与使用以下 INSERT 语句插入的信息相似的数据:

```
CREATE TABLE employees (empid BIGINT, info XML )
```

```
INSERT INTO EMPLOYEES VALUES ( 12356, '
    <department id="marketing">
        <empid>12356</empid>
        <phone>555-123-4567</phone>
     </department> ')
```

在使用上述表和信息的情况下，此 SELECT 语句将返回以下电话号码信息:

```
<phone_list><phone>555-123-4567</phone></phone_list>
```

内联型 SQL 函数和编译型 SQL 函数

SQL 函数有以下两种实现类型：内联型 SQL 函数和编译型 SQL 函数。

内联型 SQL 函数

内联型 SQL 函数是通过将 CREATE FUNCTION 语句与一个主体（该主体是一个 RETURN 语句或者内联型复合语句）配合使用而创建的 SQL 函数。内联型复合语句是使用 BEGIN ATOMIC 和 END 关键字定义的。

内联型 SQL 函数可以包含 SQL 语句和内联 SQL PL 语句（它们是 SQL PL 语句的子集）。

编译型 SQL 函数

编译型 SQL 函数是通过将 CREATE FUNCTION 语句与一个主体（该主体是一个 RETURN 语句或者编译型复合语句）配合使用而创建的 SQL 函数。编译型复合语句是使用 BEGIN 和 END 关键字定义的。

当省略了 ATOMIC 子句时，将编译 SQL 函数，这样它可以包含或引用比内联型 SQL 函数更多的 SQL PL 功能部件。编译型 SQL 函数可以包含内联型 SQL 函数中不支持的下列功能部件:

- SQL PL 语句，其中包括:
  - CASE 语句
  - REPEAT 语句
- 游标处理
- 动态 SQL
- 条件处理程序

外部例程

外部例程中的 XML 数据类型支持

使用下列编程语言编写的外部过程和函数支持 XML 数据类型的参数和变量:

- C
- C++
- COBOL
- Java
- .NET CLR 语言

外部 OLE 和 OLEDB 例程不支持数据类型为 XML 的参数。

XML 数据类型值在外部例程代码中的表示方式与 CLOB 数据类型的表示方式相同。

当声明数据类型为 XML 的外部例程参数时，将用来在数据库中创建例程的 CREATE PROCEDURE 和 CREATE FUNCTION 语句必须指定要将 XML 数据类型作为 CLOB 数据类型来存储。CLOB 值的大小应该与由 XML 参数表示的 XML 文档的大小比较接近。

以下 CREATE PROCEDURE 语句显示通过 C 编程语言并使用 XML 参数 parm1 实现的外部过程的 CREATE PROCEDURE 语句:

```
CREATE PROCEDURE myproc(IN parm1 XML AS CLOB(2M), IN parm2 VARCHAR(32000))
 LANGUAGE C
 FENCED   PARAMETER STYLE SQL     EXTERNAL NAME 'mylib!myproc';
```

在创建外部 UDF 时存在类似的注意事项，如以下示例中所示:

```
CREATE FUNCTION myfunc (IN parm1 XML AS CLOB(2M))
RETURNS SMALLINT
 LANGUAGE C
PARAMETER STYLE SQL     DETERMINISTIC
NOT FENCED
NULL CALL
NO SQL
NO EXTERNAL ACTION
EXTERNAL NAME 'mylib1!myfunc'
```

将 XML 数据作为 IN、OUT 或 INOUT 参数传递至存储过程时，将具体化该数据。如果使用的是 Java 存储过程，那么可能需要根据 XML 参数的数量和大小以及正在同时执行的外部存储过程的数目来增大堆大小（JAVA_HEAP_SZ 配置参数）。

在外部例程代码中，将按照数据库应用程序中的相同方式来访问、设置和修改 XML 参数和变量值。

为 Java 例程指定驱动程序

开发和调用 Java 例程需要指定 JDBC 或 SQLJ 驱动程序。Java 例程可以使用下面两个驱动程序之一:

- IBM 数据服务器 JDBC 和 SQLJ 驱动程序
- DB2 2 类驱动程序

缺省情况下，DB2 使用 IBM 数据服务器 JDBC 和 SQLJ 驱动程序。首选此驱动程序，它是更健壮的驱动程序；建议不要使用 DB2 2 类驱动程序，原因是当 Java 例程包含下列各项时，它就是必备驱动程序:

- 数据类型为 XML 的参数
- 数据类型为 XML 的变量
- 对 XML 数据的引用
- 对 XML 函数的引用
- 任何其他本机 XML 功能部件

如果对现有 Java 例程进行升级时发生问题，那么可以将 DB2 环境变量 DB2_USE_DB2JCCT2_JROUTINE 设置为值 NO，以使用合法的 IBM DB2 2 类驱动程序。这可通过从 DB2 命令窗口发出以下命令来实现:

```
db2set DB2_USE_DB2JCCT2_JROUTINE=NO
```

发出此命令之后，必须停止 DB2 实例，然后重新启动它才能使更改生效。

示例: Java（JDBC）过程中的 XML 和 XQuery 支持

一旦您了解了 Java 过程的基础知识、使用 Java 语言并借助 JDBC 应用程序编程接口（API）来编程以及 XQuery，就可以开始创建并使用可用来查询 XML 数据的 Java 过程。

此 Java 过程示例说明了:

- 参数样式 JAVA 过程的 CREATE PROCEDURE 语句
- 参数样式 JAVA 过程的源代码
- 数据类型 XML 的输入和输出参数
- 在查询中使用 XML 输入参数

- 将 XQuery 的结果（一个 XML 值）作为输出参数的赋值
- 将 SQL 语句的结果（一个 XML 值）作为输出参数的赋值

先决条件

在使用此 Java 过程示例之前，您可能需要阅读下列主题:

- Java 例程
- 例程
- 构建 Java 例程代码

下列示例使用一个名为 xmlDataTable 的表，该表的定义如下所示并且它包含数据:

```
CREATE TABLE xmlDataTable
(
        num INTEGER,
        xdata XML
  )@

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                                <type>car</type>
                                <make>Pontiac</make>
                                <model>Sunfire</model>
                                </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                                <type>car</type>
                                <make>Mazda</make>
                                <model>Miata</model>
                                </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Mary</name>
                                <town>Vancouver</town>
                                <street>Waterside</street>
                                </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Mark</name>
                                <town>Edmonton</town>
                                <street>Oak</street>
                                </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                                <type>animal</type>
                                <name>dog</name>
                                </doc>' PRESERVE WHITESPACE)),
        (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                                <type>car</type>
                                <make>Ford</make>
                                <model>Taurus</model>
                                </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Kim</name>
                                <town>Toronto</town>
                                <street>Elm</street>
                                </doc>' PRESERVE WHITESPACE)),
   (9, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Bob</name>
                                <town>Toronto</town>
                                <street>Oak</street>
                                </doc>' PRESERVE WHITESPACE)),
   (10, XMLPARSE(DOCUMENT '<doc>
```

```
<type>animal</type>
<name>bird</name>
</doc>' PRESERVE WHITESPACE))@
```

过程 在创建您自己的 Java 过程时，可将以下示例作为参考:

- 表 37
- 第 236 页的表 38

Java 外部代码文件

该示例说明了一个 Java 过程实现。该示例由两部分组成：过程的 CREATE PROCEDURE 语句和外部 Java 代码实现，可以根据该过程构建相关联的 Java 类。

包含下列示例的过程实现的 Java 源文件名为 `stpclass.java`，该文件包含在一个名为 `myJAR` 的 JAR 文件中。该文件具有以下格式:

表 37. Java 外部代码文件格式

```
using System;
  import java.lang.*;
  import java.io.*;
  import java.sql.*;
  import java.util.*;
  import com.ibm.db2.jcc.DB2Xml;

    public class stpclass
 {           ...
    // Java procedure implementations
            ...
 }
```

在文件顶部指示了 Java 类文件导入。如果文件中的任何过程包含将使用的 XML 类型的参数或变量，那么 `com.ibm.db2.jcc.DB2Xml` 导入是必需的。

一定要记下类文件的名称和包含给定过程实现的 JAR 名称。这些名称很重要，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 在运行时能够找到该类。

示例 1：使用 XML 参数的参数样式 JAVA 过程

此示例显示下列各项内容:

- 参数样式 JAVA 过程的 CREATE PROCEDURE 语句
- 使用 XML 参数的参数样式 JAVA 过程的 Java 代码

此过程采用输入参数 `inXML`，将包含该值的行插入表中，使用 SQL 语句和 XQuery 表达式来 queriesXML 数据，设置两个输出参数 `outXML1` 和 `outXML2`。

表 38. 用来创建使用 XML 参数的参数样式 JAVA 过程的代码

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT out1XML XML as CLOB (1K),
                               OUT out2XML XML as CLOB (1K)
                                  )
  DYNAMIC RESULT SETS 0
  DETERMINISTIC
   LANGUAGE JAVA
 PARAMETER STYLE JAVA     MODIFIES SQL DATA
   FENCED    THREADSAFE
  DYNAMIC RESULT SETS 0
  PROGRAM TYPE SUB
  NO DBINFO
EXTERNAL NAME 'myJar:stpclass.xmlProc1'@
```

表 38. 用来创建使用 XML 参数的参数样式 JAVA 过程的代码 (续)

```
//*************************************************************************
//  Stored Procedure: XMLPROC1
//
//  Purpose:   Inserts XML data into XML column; queries and returns XML data
//
//  Parameters:
//
//   IN:      inNum -- the sequence of XML data to be insert in xmldata table
//            inXML -- XML data to be inserted
//   OUT:     out1XML -- XML data to be returned
//            out2XML -- XML data to be returned
//
//*************************************************************************
public void xmlProc1(int inNum,
                                            DB2Xml inXML ,
                                            DB2Xml[] out1XML,
                                            DB2Xml[] out2XML
                    )
  throws Exception
{
       Connection con = DriverManager.getConnection("jdbc:default:connection");

       // Insert data including the XML parameter value into a table
       String query = "INSERT INTO xmlDataTable (num, inXML ) VALUES ( ?, ? )" ;
       String xmlString = inXML.getDB2String() ;

       stmt = con.prepareStatement(query);
       stmt.setInt(1, inNum);
       stmt.setString (2, xmlString );
       stmt.executeUpdate();
       stmt.close();

       // Query and retrieve a single XML value from a table using SQL
       query = "SELECT xdata from xmlDataTable WHERE num = ? " ;

       stmt = con.prepareStatement(query);
       stmt.setInt(1, inNum);
       ResultSet rs = stmt.executeQuery();

                if ( rs.next() )
       {  out1Xml[0] = (DB2Xml) rs.getObject(1);  }

             rs.close();
       stmt.close();

       // Query and retrieve a single XML value from a table using XQuery
       query = "XQUERY for $x in db2-fn:xmlcolumn(\"xmlDataTable.xdata\")/doc
                                where $x/make = \'Mazda\'
                                return <carInfo>{$x/make}{$x/model}</carInfo>";

       stmt = con.createStatement();

                rs = stmt.executeQuery( query );

                if ( rs.next() )
                {  out2Xml[0] = (DB2Xml) rs.getObject(1) ;  }

             rs.close();
       stmt.close();
       con.close();

             return ;
}
```

示例：C# .NET CLR 过程中的 XML 和 XQuery 支持

一旦您了解了过程的基础知识、.NET 通用语言运行时例程的实质、XQuery 和 XML，就可以开始创建 CLR 过程并将它们与 XML 功能部件配合使用。

以下示例说明了参数类型为 XML 的 C# .NET CLR 过程以及如何更新和查询 XML 数据。

先决条件

在使用该 CLR 过程示例之前，您可能需要阅读下列概念主题：

- .NET 通用语言运行时（CLR）例程
- 从 DB2 命令窗口创建 .NET CLR 例程
- 使用例程的优点
- 《开发 ADO.NET 和 OLE DB 应用程序》中的『构建通用语言运行时（CLR）.NET 例程』

下列示例使用一个名为 xmlDataTable 的表，该表的定义如下所示：

```
CREATE TABLE xmlDataTable
(
        num INTEGER,
        xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                               <type>car</type>
                               <make>Pontiac</make>
                               <model>Sunfire</model>
                               </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                               <type>car</type>
                               <make>Mazda</make>
                               <model>Miata</model>
                               </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                               <type>person</type>
                               <name>Mary</name>
                               <town>Vancouver</town>
                               <street>Waterside</street>
                               </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                               <type>person</type>
                               <name>Mark</name>
                               <town>Edmonton</town>
                               <street>Oak</street>
                               </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                               <type>animal</type>
                               <name>dog</name>
                               </doc>' PRESERVE WHITESPACE)),
        (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                               <type>car</type>
                               <make>Ford</make>
                               <model>Taurus</model>
                               </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                               <type>person</type>
                               <name>Kim</name>
                               <town>Toronto</town>
                               <street>Elm</street>
                               </doc>' PRESERVE WHITESPACE)),
```

```
(9, XMLPARSE(DOCUMENT '<doc>
                       <type>person</type>
                       <name>Bob</name>
                       <town>Toronto</town>
                       <street>Oak</street>
                       </doc>' PRESERVE WHITESPACE)),
(10, XMLPARSE(DOCUMENT '<doc>
                        <type>animal</type>
                        <name>bird</name>
                        </doc>' PRESERVE WHITESPACE))@
```

过程 在创建您自己的 C# CLR 过程时，可将下列示例作为参考:

- C# 外部代码文件
- 示例 1: 带有 XML 功能部件的 C# 参数样式 GENERAL 过程

C# 外部代码文件

该示例由两部分组成: 过程的 CREATE PROCEDURE 语句和外部 C# 代码实现，可以根据该过程来构建相关联的组合件。

包含下列示例的过程实现的 C# 源文件名为 gwenProc.cs，并且具有以下格式:

表 39. C# 外部代码文件格式

```
using System;
  using System.IO;
  using System.Data;
  using IBM.Data.DB2;
  using IBM.Data.DB2Types;

  namespace bizLogic
{
        class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

在文件顶部指示了文件包含的内容。如果文件中的任何过程包含 SQL，那么必须包含 IBM.Data.DB2。如果文件中的任何过程包含 XML 类型的参数或变量，那么必须包含 IBM.Data.DB2Types。此文件中有一个名称空间声明和一个包含过程的 empOps 类。是否使用名称空间是可选的。如果使用了名称空间，那么名称空间必须出现在 CREATE PROCEDURE 语句的 EXTERNAL 子句中所提供的组合件路径名中。

一定要记下类文件的名称、名称空间以及包含给定过程实现的类名。这些名称是很重要的，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 可以找到 CLR 过程的组合件和类。

示例 1: 使用 XML 功能部件的 C# 参数样式 GENERAL 过程

此示例显示下列各项内容:

- 参数样式 GENERAL 过程的 CREATE PROCEDURE 语句
- 使用 XML 参数的参数样式 GENERAL 过程的 C# 代码

此过程采用两个参数，一个整数 inNum 和一个 inXML。这些值被插入到 xmlDataTable 表中。然后使用 XQuery 来检索 XML 值。使用 SQL 来检索另一个 XML 值。将检

索到的 XML 值指定给两个输出参数：outXML1 和 outXML2。不会返回任何结果集。

表 40. 用来创建 C# 参数样式 GENERAL 过程的代码

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K)
                             )
  LANGUAGE CLR
  PARAMETER STYLE GENERAL
  DYNAMIC RESULT SETS 0
FENCED
THREADSAFE
DETERMINISTIC
NO DBINFO
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!xmlProc1' ;
```

```
//*************************************************************************
//  Stored Procedure: xmlProc1
//
//  Purpose:   insert XML data into XML column
//
//  Parameters:
//
//   IN:      inNum -- the sequence of XML data to be insert in xmldata table
//            inXML -- XML data to be inserted
//   OUT:   outXML1 -- XML data returned - value retrieved using XQuery
//          outXML2 -- XML data returned - value retrieved using SQL
//*************************************************************************
```

表 40. 用来创建 C# 参数样式 GENERAL 过程的代码 (续)

```
      public static void xmlProc1 (     int      inNum, DB2Xml   inXML,
                              out DB2Xml   outXML1, out DB2Xml   outXML2 )
      {
         // Create new command object from connection context
                  DB2Parameter parm;
                  DB2Command cmd;
                  DB2DataReader reader = null;
                  outXML1 = DB2Xml.Null;
                  outXML2 = DB2Xml.Null;

         // Insert input XML parameter value into a table
         cmd = DB2Context.GetCommand();
                  cmd.CommandText = "INSERT INTO "
                           + "xmlDataTable( num , xdata ) "
                           + "VALUES( ?, ? )";
;

         parm = cmd.Parameters.Add("@num", DB2Type.Integer );
         parm.Direction = ParameterDirection.Input;
         cmd.Parameters["@num"].Value = inNum;
         parm = cmd.Parameters.Add("@data", DB2Type.Xml);
         parm.Direction = ParameterDirection.Input;
                  cmd.Parameters["@data"].Value = inXML ;
         cmd.ExecuteNonQuery();
         cmd.Close();

         // Retrieve XML value using XQuery
                        and assign value to an XML output parameter
         cmd = DB2Context.GetCommand();
         cmd.CommandText = "XQUERY for $x " +
                           "in db2-fn:xmlcolumn(\"xmlDataTable.xdata\")/doc "+
                           "where $x/make = \'Mazda\' " +
                           "return <carInfo>{$x/make}{$x/model}</carInfo>";
         reader = cmd.ExecuteReader();
         reader.CacheData= true;

         if (reader.Read())
         {  outXML1 = reader.GetDB2Xml(0);  }
         else
         {  outXML1 = DB2Xml.Null;  }

         reader.Close();
         cmd.Close();

         // Retrieve XML value using SQL
                        and assign value to an XML output parameter value
         cmd = DB2Context.GetCommand();
         cmd.CommandText = "SELECT xdata "
                            + "FROM xmlDataTable "
                            + "WHERE num = ?";

         parm = cmd.Parameters.Add("@num", DB2Type.Integer );
         parm.Direction = ParameterDirection.Input;
         cmd.Parameters["@num"].Value = inNum;
         reader = cmd.ExecuteReader();
         reader.CacheData= true;

         if (reader.Read())
         {  outXML2 = reader.GetDB2Xml(0);  }
         else
         {  outXML = DB2Xml.Null;  }

         reader.Close();
         cmd.Close();

         return;
      }
```

示例：C 过程中的 XML 和 XQuery 支持

一旦您了解了过程的基础知识、C 例程的实质、XQuery 和 XML，就可以开始创建 C 过程并将它们与 XML 功能部件配合使用。

以下示例说明了参数类型为 XML 的 C 过程以及如何更新和查询 XML 数据。

先决条件

在使用该 C 过程示例之前，您可能需要阅读以下概念主题：

- 使用例程的优点

下列示例使用一个名为 xmlDataTable 的表，该表的定义如下所示：

```
CREATE TABLE xmlDataTable
(
        num INTEGER,
        xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                               <type>car</type>
                               <make>Pontiac</make>
                               <model>Sunfire</model>
                               </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                               <type>car</type>
                               <make>Mazda</make>
                               <model>Miata</model>
                               </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                               <type>person</type>
                               <name>Mary</name>
                               <town>Vancouver</town>
                               <street>Waterside</street>
                               </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                               <type>person</type>
                               <name>Mark</name>
                               <town>Edmonton</town>
                               <street>Oak</street>
                               </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                               <type>animal</type>
                               <name>dog</name>
                               </doc>' PRESERVE WHITESPACE)),
        (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                               <type>car</type>
                               <make>Ford</make>
                               <model>Taurus</model>
                               </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                               <type>person</type>
                               <name>Kim</name>
                               <town>Toronto</town>
                               <street>Elm</street>
                               </doc>' PRESERVE WHITESPACE)),
   (9, XMLPARSE(DOCUMENT '<doc>
                               <type>person</type>
                               <name>Bob</name>
                               <town>Toronto</town>
                               <street>Oak</street>
                               </doc>' PRESERVE WHITESPACE)),
```

```
(10, XMLPARSE(DOCUMENT '<doc>
                           <type>animal</type>
                           <name>bird</name>
                           </doc>' PRESERVE WHITESPACE))
```

过程 在创建您自己的 C 过程时，可将下列示例作为参考:

- C 外部代码文件
- 示例 1: 带有 XML 功能部件的 C 参数样式 SQL 过程

C 外部代码文件

该示例由两部分组成: 过程的 CREATE PROCEDURE 语句和外部 C 代码实现，可以根据该过程来构建相关联的组合件。

包含下列示例的过程实现的 C 源文件名为 gwenProc.SQC，并且具有以下格式:

表 41. C 外部代码文件格式

```
 #include <stdio.h>
#include <string.h>
#include <stdlib.h>
 #include <sqlda.h>
 #include <sqlca.h>
 #include <sqludf.h>
 #include <sql.h>
 #include <memory.h>

// C procedures
               ...
```

在文件顶部指示了文件包含的内容。嵌入式 SQL 例程中的 XML 支持不需要额外的包含文件。

一定要记下文件的名称以及与过程实现相对应的函数的名称。这些名称是很重要的，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 数据库管理器可以找到与 C 过程相对应的库和入口点。

示例 1: 使用 XML 功能部件的 C 参数样式 SQL 过程

此示例显示下列各项内容:

- 参数样式 SQL 过程的 CREATE PROCEDURE 语句
- 使用 XML 参数的参数样式 SQL 过程的 C 代码

此过程将接收两个输入参数。第一个输入参数名为 inNum，其类型为 INTEGER。第二个输入参数名为 inXML，其类型为 XML。这些输入参数的值用来将一行插入到 xmlDataTable 表中。然后使用 SQL 语句来检索 XML 值。使用 XQuery 表达式来检索另一个 XML 值。将检索到的 XML 值分别指定给两个输出参数: out1XML 和 out2XML。不会返回任何结果集。

表 42. 用来创建 C 参数样式 SQL 过程的代码

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                             IN inXML  XML as CLOB (1K),
                             OUT inXML  XML as CLOB (1K),
                             OUT inXML  XML as CLOB (1K)
                           )
LANGUAGE C
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
THREADSAFE
DETERMINISTIC
NO DBINFO
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc!xmlProc1' ;
```

```
//*************************************************************************
//  Stored Procedure: xmlProc1
//
//  Purpose:   insert XML data into XML column
//
//  Parameters:
//
//   IN:    inNum -- the sequence of XML data to be insert in xmldata table
//          inXML -- XML data to be inserted
//   OUT:   out1XML -- XML data returned - value retrieved using XQuery
//          out2XML -- XML data returned - value retrieved using SQL
//*************************************************************************
```

表 42. 用来创建 C 参数样式 SQL 过程的代码 (续)

```
#ifdef __cplusplus
extern "C"
#endif
SQL_API_RC SQL_API_FN testSecA1(sqlint32* inNum,
                                SQLUDF_CLOB* inXML,
                                SQLUDF_CLOB* out1XML,
                                SQLUDF_CLOB* out2XML,
                                SQLUDF_NULLIND *inNum_ind,
                                SQLUDF_NULLIND *inXML_ind,
                                SQLUDF_NULLIND *out1XML_ind,
                                SQLUDF_NULLIND *out2XML_ind,
                                SQLUDF_TRAIL_ARGS)
{
  char *str;
  FILE *file;

  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
       sqlint32 hvNum1;
       SQL TYPE IS XML AS CLOB(200) hvXML1;
       SQL TYPE IS XML AS CLOB(200) hvXML2;
       SQL TYPE IS XML AS CLOB(200) hvXML3;
  EXEC SQL END DECLARE SECTION;

  /* Check null indicators for input parameters */
  if ((*inNum_ind < 0) ||  (*inXML_ind < 0)) {
     strcpy(sqludf_sqlstate, "38100");
     strcpy(sqludf_msgtext, "Received null input");
     return 0;
  }

  /* Copy input parameters to host variables */
  hvNum1 = *inNum;
  hvXML1.length = inXML->length;
  strncpy(hvXML1.data, inXML->data, inXML->length);

  /* Execute SQL statement */
  EXEC SQL
     INSERT INTO xmlDataTable (num, xdata) VALUES (:hvNum1, :hvXML1);

  /* Execute SQL statement */
  EXEC SQL
     SELECT xdata INTO :hvXML2
        FROM xmlDataTable
           WHERE num = :hvNum1;

  sprintf(stmt5, "SELECT XMLQUERY('for $x in $xmldata/doc
                                 return <carInfo>{$x/model}</carInfo>'
                                 passing by ref xmlDataTable.xdata
                                     as \"xmldata\" returning sequence)
                  FROM xmlDataTable WHERE num = ?");

  EXEC SQL PREPARE selstmt5 FROM :stmt5 ;
  EXEC SQL DECLARE c5 CURSOR FOR selstmt5;
  EXEC SQL OPEN c5 using :hvNum1;
  EXEC SQL FETCH c5 INTO :hvXML3;

  exit:

  /* Set output return code */
  *outReturnCode = sqlca.sqlcode;
  *outReturnCode_ind = 0;

  return 0;
}
```

例程的性能

例程的性能会受到各种因素的影响，这些因素包括：例程的类型和实现、例程中包含的 SQL 语句数目、例程中 SQL 的复杂程度、例程的参数数目、例程实现中的逻辑的效率以及例程中的错误处理方法等等。因为用户通常选择通过实现例程来提高应用程序的性能，所以，获得最佳例程性能是很重要的。

下表概括了会影响例程性能的一些一般因素，并对如何通过改变每个因素来提高例程性能提供了建议。有关会影响特定例程类型的性能因素的更多详细信息，请参阅关于特定例程类型的性能和调整主题。

表 43. 性能注意事项和关于例程性能的建议

性能注意事项	性能建议
例程类型：过程、函数和方法	• 过程、函数和方法的用途不同，引用的位置也不同。由于它们在功能上存在差异，因此很难直接比较它们的性能。 • 一般来说，有时候可以将过程改写为函数（尤其是在过程返回标量值并且仅查询数据的情况下），这样对性能稍微有所改善。但是，通常是通过简化实现 SQL 逻辑所需要的 SQL 来改善性能的。 • 用户定义的初始化过程很复杂的函数可以利用暂存区来存储第一次调用时所需要的任何值，以便在后续调用中也可以使用这些值。
例程实现：系统定义的例程或用户定义的例程	• 对于等价的逻辑，内置例程的执行性能最佳，其次是系统定义的例程。这是因为与用户定义的例程相比，这些例程与数据库引擎的关系更紧密。 • 如果用户定义的例程的代码编写得很好并且遵循最佳实践，那么它们的执行性能也会很好。
例程实现：SQL 例程或外部例程实现	• 与外部例程相比，SQL 例程的效率更高，因为它们是直接由 DB2 数据库服务器执行的。 • SQL 过程通常比逻辑上等价的外部过程的执行性能更好。 • 对于简单逻辑，SQL 函数的性能将与等价的外部函数的性能相当。 • 对于复杂逻辑（例如，需要使用少量 SQL 的数学算法和字符串处理函数），最好是使用采用低级编程语言（例如，C 语言）编写的外部例程。这是因为这种外部例程对 SQL 支持的依赖性较低。 • 有关受支持的外部例程编程语言选项的特征（包括性能）的比较，请参阅例程实现之间的比较。

表 43. 性能注意事项和关于例程性能的建议 (续)

性能注意事项	性能建议
外部例程实现编程语言	• 有关在选择外部例程实现时应该考虑的性能特点的比较，请参阅外部例程 API 和编程语言的比较。 • Java（JDBC 和 SQLJ API） – 在创建内存需求非常大的 Java 例程时，最好是指定 FENCED NOT THREADSAFE 子句。而对于具有平均内存需求的 Java 例程可以指定 FENCED THREADSAFE 子句。 – 对于受防护的线程安全 Java 例程调用，DB2 将尝试选择一个线程化的 Java 受防护方式进程，它具有足够大的 Java 堆来运行例程。如果大型堆使用者在它们自己的进程中未能进行隔离，那么在多线程 Java db2fmp 进程中会导致“用尽 Java 堆”错误。然而，FENCED THREADSAFE 例程的执行性能就很好，这是因为这些例程之间会共享少量的 JVM。 • C 和 C++ – 通常，C 和 C++ 例程比其他外部例程实现和 SQL 例程的执行性能更好。 – 为了获得最佳执行性能，如果 C 和 C++ 例程将被部署到 32 位 DB2 实例，那么应按 32 位格式来编译这些例程；而如果要将它们部署到 64 位 DB2 实例，那么应按 64 位格式来编译它们。 • COBOL – 通常，COBOL 例程的执行性能很好，但是建议不要将 COBOL 作为例程实现。
例程中的 SQL 语句数目	• 例程中应包含多个 SQL 语句，否则，例程调用的开销将不能获得很好的性能成本效益。 • 能够完成下列任务的逻辑是最适合于例程封装的逻辑：必须执行多次数据库查询，处理中间结果，最终返回已使用的一部分数据。复杂数据挖掘和需要查询相关数据的大型更新就是这种逻辑的一些示例。大量的 SQL 处理是在数据库服务器上完成的，并且只将较小的数据结果集返回给调用者。
例程中的 SQL 语句的复杂性	• 将非常复杂的查询包含在例程中是很有意义的，这样就可以充分利用数据库服务器的内存和性能优势。 • 不必担心 SQL 语句过于复杂。
例程中的静态 SQL 或动态 SQL 的执行情况	• 通常，静态 SQL 比动态 SQL 的执行性能更好。在例程中，当您使用静态 SQL 或动态 SQL 时没有其他差别。
例程的参数数目	• 将例程的参数数目减少到最低程度可以提高例程的性能，因为这样做可使得在例程和例程调用程序之间传递的缓冲区数目减少到最低程度。

表 43. 性能注意事项和关于例程性能的建议 (续)

性能注意事项	性能建议
例程参数的数据类型	• 可以通过在例程定义中使用 VARCHAR 参数（而不是 CHAR 参数）来提高例程的性能。通过使用 VARCHAR 数据类型而不是 CHAR 数据类型，可以防止 DB2 在传递参数之前为参数填充空格，并且缩短在网络中传输该参数所需要的时间。 例如，如果客户机应用程序将字符串 ″A SHORT STRING″ 传递给一个期望使用 CHAR(200) 参数的例程，那么 DB2 必须为参数填充 186 个空格，并以 NULL 终止该字符串，然后将由 200 个字符组成的整个字符串和 NULL 终止符通过网络发送至例程。 然而，如果将同一字符串 ″*A SHORT STRING*″ 传递给期望使用 VARCHAR(200) 参数的例程，那么 DB2 只会通过网络来传递由这 14 个字符组成的字符串和 NULL 终止符。
例程参数的初始化	• 最好是始终对例程的输入参数进行初始化，尤其是在输入例程参数值为空的情况下。当例程参数值为空时，可以将更短的或空的缓冲区（而不是整个缓冲区）传递给例程，这样做可以提高性能。
例程中的局部变量数	• 将例程中已声明的局部变量数减少到最低程度，就会使例程中执行的 SQL 语句数减少到最低程度，从而提高性能。 • 通常，应尽量少使用变量。如果复用变量不会造成语义上发生混淆，那么复用变量。
例程中局部变量的初始化	• 如果有可能，就最好是使用单个 SQL 语句来初始化多个局部变量，这样做可以缩短例程的 SQL 执行总时间。
由例程返回的结果集的数目	• 如果可以减少由例程返回的结果集的数目，那么可以提高例程的性能。
由例程返回的结果集的大小	• 确保对于由例程返回的每个结果集，用于定义结果的查询将尽可能多地过滤返回的列和返回的行数。返回不需要的数据列或行将使得效率不高，并且可能会导致例程性能不是最佳。
例程中逻辑的效率	• 与任何应用程序一样，例程的性能会受到实现方法不佳的算法的限制。追求的目标是尽可能高效率地编写例程，并且尽可能应用通常建议的编码最佳实践。 • 分析 SQL，并尽可能减少对最简单的 SQL 形式的查询。这通常可以使用以下方法来实现：使用 CASE 表达式而不使用 CASE 语句；或者将多个 SQL 语句折叠成单个语句，并使用一个 CASE 表达式作为开关。

表 43. 性能注意事项和关于例程性能的建议（续）

性能注意事项	性能建议
例程的运行时方式（FENCED 或 NOT FENCED 子句规范）	**NOT FENCED 子句的用途:** • 一般来说，使用 NOT FENCED 子句来创建例程优于使用 FENCED 子句来创建例程。前一种方法会使例程与 DB2 数据库管理器在同一进程中运行，而后一种方法会使例程在引擎的地址空间之外的一个特定的 DB2 进程中运行。 • 虽然您可能认为运行不受防护的例程会提高例程的性能，但是不受防护的例程中的用户代码可能会有意或无意地毁坏数据库或破坏数据库控制结构。因此，仅当您需要获取最佳性能并且确信例程足够安全的情况下，才应使用 NOT FENCED 子句。（有关评定和降低将 C/C++ 例程注册为 NOT FENCED 例程的风险的信息，请参阅例程的安全性。如果例程在数据库管理器的进程中运行时不是足够安全，那么应使用 FENCED 子句来创建例程。为了限制创建和运行可能不安全的代码，DB2 要求用户具有特殊特权 CREATE_NOT_FENCED_ROUTINE 才能创建 NOT FENCED 例程。 • 如果您在运行 NOT FENCED 例程时发生异常终止，那么如果例程被注册为 NO SQL，数据库管理器就会尝试进行相应的恢复。但是，对于未定义为 NO SQL 的例程，数据库管理器将失败。 • 如果 NOT FENCED 例程使用 GRAPHIC 或 DBCLOB 数据，那么必须使用 WCHARTYPE NOCONVERT 选项来预编译这些例程。

表 43. 性能注意事项和关于例程性能的建议 (续)

性能注意事项	性能建议
例程的运行时方式（FENCED 或 NOT FENCED 子句规范）	**FENCED THREADSAFE 子句的用途** • 使用 FENCED THREADSAFE 子句创建的例程与其他例程运行于同一进程中。更准确地说，非 Java 例程共享同一个进程，而 Java(TM) 例程共享另一个进程，与使用其他语言编写的例程分隔开。通过这种分隔，可以使 Java 例程不会受到使用其他语言编写的可能会产生很多错误的例程的影响。另外，Java 例程的进程包含 JVM，这将导致内存成本很高，并且其他例程类型不会使用该 JVM。多次调用 FENCED THREADSAFE 例程时会共享资源，因此，这种例程比 FENCED NOT THREADSAFE 例程的系统开销少，因为每个 FENCED NOT THREADSAFE 例程都在它们自己的专用进程中运行。 • 如果您认为让某一例程与其他例程在同一进程中运行是足够安全的，那么在注册该例程时可以使用 THREADSAFE 子句。与 NOT FENCED 例程一样，在“例程的安全性注意事项”这一主题中提供了有关评定和降低将 C/C++ 例程注册为 FENCED THREADSAFE 例程的风险的信息。 • 如果 FENCED THREADSAFE 例程异常结束，那么只有正在运行此例程的线程才会终止。进程中的其他例程将继续运行。但是，导致此线程异常结束的故障又会影响进程中的其他例程线程，将导致它们捕获、挂起或破坏数据。当一个线程异常终止之后，进程将不再用于新的例程调用。一旦所有活动用户都在此进程中完成了他们的作业，此进程就会终止。 • 当注册 Java 例程时，除非您另有声明，否则都认为它们是 THREADSAFE 的。缺省情况下，所有其他 LANGUAGE 类型是 NOT THREADSAFE。不能将使用 LANGUAGE OLE 和 OLE DB 的例程指定为 THREADSAFE。 • NOT FENCED 例程必须是 THREADSAFE 的。不能将例程注册为 NOT FENCED NOT THREADSAFE（SQLCODE -104）。 • 在 UNIX(R) 上，用户可以通过查找 db2fmp（Java）或 db2fmp（C）来查看他们的 Java 和 C THREADSAFE 进程。
例程的运行时方式（FENCED 或 NOT FENCED 子句规范）	**FENCED NOT THREADSAFE 方式** • 每个 FENCED NOT THREADSAFE 例程都在它们自己的专用进程中运行。如果您在运行许多个例程，那么此方式可能对数据库系统的性能造成不利影响。如果某一例程与其他例程在同一进程中运行时不是足够安全，那么在注册该例程时可以使用 NOT THREADSAFE 子句。 • 在 UNIX 上，NOT THREADSAFE 进程作为 db2fmp (pid) 出现（其中 pid 是使用受防护方式进程的代理程序的进程标识），或者对于合用的 NOT THREADSAFE db2fmp 作为 db2fmp (idle) 出现。

表 43. 性能注意事项和关于例程性能的建议 (续)

性能注意事项	性能建议
例程中的 SQL 访问级别：NO SQL、CONTAINS SQL、READS SQL DATA 和 MODIFIES SQL DATA	• 使用较低级别的 SQL 访问子句创建的例程，将比使用较高级别的 SQL 访问子句创建的例程的执行性能更高。因此，应该使用最受限制级别的 SQL 访问子句来声明例程。例如，如果例程只读取 SQL 数据，那么不要使用 MODIFIES SQL DATA 子句创建它，而是使用受到更多限制的 READS SQL DATA 子句来创建它。
例程决定论（DETERMINISTIC 或 NOT DETERMINISTIC 子句规范）	• 使用 DETERMINISTIC 或 NOT DETERMINISTIC 子句来声明例程对例程性能没有影响。
例程执行的外部操作的数目和复杂性（EXTERNAL ACTION 子句规范）	• 根据外部例程执行的外部操作的数目和复杂性，可能会影响例程的性能。影响例程性能的因素包括：网络流量、对文件的读写访问权、执行外部操作所需要的时间以及与外部操作代码中的挂起或行为相关的风险。
当输入参数为空时的例程调用（CALLED ON NULL INPUT 子句规范）	• 如果接收到空输入参数值时导致不执行任何逻辑并且被例程立即返回，那么可以修改例程，以便在检测到空输入参数值时不对它进行完全调用。要创建一个只要接收到例程输入参数就提前结束调用的例程，在创建该例程时指定 CALLED ON NULL INPUT 子句即可。
XML 类型的过程参数	• 在使用 C 或 JAVA 编程语言实现的外部过程中传递 XML 数据类型的参数，比在 SQL 过程中传递时的效率低很多。当传递一个或多个采用 XML 数据类型的参数时，应考虑使用 SQL 过程而不使用外部过程。 • 将 XML 数据作为 IN、OUT 或 INOUT 参数传递至存储过程时，将具体化该数据。如果使用的是 Java 存储过程，那么可能需要根据 XML 参数的数量和大小以及正在同时执行的外部存储过程的数目来增大堆大小（JAVA_HEAP_SZ 配置参数）。

一旦创建和部署了例程，可能就很难确定是哪些特定于环境和例程的因素在影响例程的性能，因此，在设计例程时就考虑性能问题是很重要的。

样本应用程序

pureXML 样本

pureXML 功能允许将结构良好的 XML 文档以分层格式存储在表列中。XML 列使用新的 XML 数据类型来定义。由于 pureXML 功能已完全集成到 DB2 数据库系统中，因此可通过使用 DB2 功能来访问和管理存储的 XML 数据。此功能包括管理支持、应用程序开发支持以及通过对 XQuery、SQL 或者 SQL/XML 函数的组合的支持来高效搜索和检索 XML 数据。

提供了各种样本来说明 XML 支持；这些样本大致分为以下几类：

管理样本

这些样本说明了下列功能：

- XML 模式支持：XML 文档的模式注册和验证

- “为 XML 数据建立索引”支持：对 XML 值的不同节点类型建立索引
- XML 的实用程序支持：Import、export、runstats、db2look 和 db2batch 对 XML 数据类型的支持

应用程序开发样本

这些样本说明了下列功能：

- XML 插入、更新和删除：将 XML 值插入 XML 类型列，更新和删除现有值
- XML 解析、验证和序列化支持：隐式和显式解析兼容数据类型、验证 XML 文档以及序列化 XML 数据
- 混合使用 SQL 和 XQuery：使用 SQL/XML 函数（例如，XMLTABLE 和 XMLQUERY）和 XMLEXISTS 谓词
- SQL 过程和外部过程中的 XML 数据类型支持：通过包含 XML 数据类型的参数来将 XML 数据传递至 SQL 过程和外部过程
- 带注释的 XML 模式分解支持：根据带注释的 XML 模式来分解 XML 文档
- XML 发布函数：使用函数来构造 XML 值

XQuery 样本

这些样本说明如何使用以 XQuery 和 SQL/XML 编写的 AXIS、FLWOR 表达式和查询。

可在以下位置找到这些样本：

- 在 Windows 上：`%DB2PATH%\sqllib\samples\xml`（其中 `%DB2PATH%` 是一个变量，它确定 DB2 数据库服务器的安装位置）
- 在 UNIX 上：`$HOME/sqllib/samples/xml`（其中 $HOME 是实例所有者的主目录）

pureXML － 管理样本

这些样本说明了对各种管理功能的 pureXML 支持，包括 XML 模式支持、实用程序支持和 XML 数据建立索引支持。

采用各种编程语言提供了这些样本，并且可以在以下位置的特定于语言的子目录中找到它们：

- 在 Windows 上：`%DB2PATH%\sqllib\samples\xml`（其中 `%DB2PATH%` 是一个变量，它确定 DB2 数据库服务器的安装位置）
- 在 UNIX 上：`$HOME/sqllib/samples/xml`（其中 $HOME 是实例所有者的主目录）

表 44. XML 模式支持 － 有关模式注册、验证和兼容模式演进的样本

按语言归类的样本	样本程序名	程序描述
CLI	xsupdate.c	更新已注册的 XML 模式，以确保原始模式和新模式兼容。
C	xmlschema.sqc	向数据库注册 XML 模式，然后使用已注册的模式来验证和插入 XML 文档。

表 44. XML 模式支持 – 有关模式注册、验证和兼容模式演进的样本 (续)

按语言归类的样本	样本程序名	程序描述
CLP	xmlschema.db2	向数据库注册 XML 模式，然后使用已注册的模式来验证和插入 XML 文档。
	xsupdate.db2	更新已注册的 XML 模式，以确保原始模式和新模式兼容。
JDBC	XmlSchema.java	向数据库注册 XML 模式，然后使用已注册的模式来验证和插入 XML 文档。
	XsUpdate.java	更新已注册的 XML 模式，以确保原始模式和新模式兼容。
SQLJ	XmlSchema.sqlj	向数据库注册 XML 模式，然后使用已注册的模式来验证和插入 XML 文档。

表 45. 实用程序支持: Import、Export、runstats、db2look、reorg 和 db2batch 支持 XML 数据类型样本

按语言归类的样本	样本程序名	程序描述
C	xmlrunstats.sqc	对包含 XML 类型列的表执行 RUNSTATS。
	lobstoxml.sqc	使用 IMPORT 和 EXPORT 命令将 LOB 数据移入 XML 列
	impexpxml.sqc	导入和导出 XML 文档。
	xmlload.sqc	使用各种 LOAD 命令选项将 XML 文档装入到 DB2 表中。
CLP	xmlrunstats.db2	对包含 XML 类型列的表执行 RUNSTATS。
	xmlolic.db2	重新组织对表定义的索引以及如何重新组织范围分区表的非分区索引。
	xmldb2batch.db2	db2batch 对 XML 数据类型的支持。
	xmldb2look.db2	db2look 对 XML 数据类型的支持。
	lobstoxml.db2	使用 IMPORT 和 EXPORT 命令将 LOB 数据移入 XML 列
	impexpxml.db2	导入和导出 XML 文档。
	xmlload.db2	使用各种 LOAD 命令选项将 XML 文档装入到 DB2 表中。
JDBC	XmlRunstats.java	对包含 XML 类型列的表执行 RUNSTATS。

表 46. “为 XML 数据建立索引”支持: XML 数据的索引的样本

按语言归类的样本	样本程序名	程序描述
C	xmlindex.sqc	创建一个索引并在 XQuery 查询中使用它。
	xmlconst.sqc	使用 XML 模式创建一个具有 UNIQUE 和 VARCHAR 长度约束的索引。

表 46. “为 XML 数据建立索引”支持：XML 数据的索引的样本 (续)

按语言归类的样本	样本程序名	程序描述
CLI	xmlindex.c	创建一个索引并在 XQuery 查询中使用它。
	xmlconst.c	使用 XML 模式创建一个具有 UNIQUE 和 VARCHAR 长度约束的索引。
CLP	xmlindex.db2	创建一个索引并在 XQuery 查询中使用它。
	xmlconst.db2	使用 XML 模式创建一个具有 UNIQUE 和 VARCHAR 长度约束的索引。
JDBC	XmlIndex.java	创建一个索引并在 XQuery 查询中使用它。
	XmlConst.java	使用 XML 模式创建一个具有 UNIQUE 和 VARCHAR 长度约束的索引。
SQLJ	XmlIndex.sqlj	创建一个索引并在 XQuery 查询中使用它。
	XmlConst.sqlj	使用 XML 模式创建一个具有 UNIQUE 和 VARCHAR 长度约束的索引。

pureXML – 应用程序开发样本

这些样本说明了诸如下列应用程序开发功能的 XML 支持：插入、更新和删除、XML 解析、验证、序列化、混合使用 SQL/XML、SQL 和外部存储过程中的 XML 数据类型支持、XML 分解以及 SQL/XML 发布函数。

采用各种编程语言提供了这些样本，并且可以在以下位置的特定于语言的子目录中找到它们：

- 在 Windows 上：%DB2PATH%\sqllib\samples\xml（其中 %DB2PATH% 是一个变量，它确定 DB2 数据库服务器的安装位置）
- 在 UNIX 上：$HOME/sqllib/samples/xml（其中 $HOME 是实例所有者的主目录）

表 47. pureXML – 应用程序开发样本

按语言归类的样本	样本程序名	程序描述
CLI	xmlinsert.c	将 XML 文档插入到 XML 数据类型的列中。
	xmlupdel.c	更新和删除表中的 XML 文档。
	xmlread.c	读取存储在表中的 XML 数据。
	reltoxmldoc.c	使用各种 SQL/XML 发布函数直接根据存储在关系表中的数据来创建 XML 文档。
	xmltotable.c	使用 SQL/XML 函数（例如，XMLTABLE 和 XMLQUERY）和 XMLEXISTS 谓词将 XML 文档中的数据插入到关系表中。
	simple_xmlproc.c	具有 XML 类型参数的简单存储过程

表 47. pureXML – 应用程序开发样本 (续)

按语言归类的样本	样本程序名	程序描述
	simple_xmlproc_client.c	要调用 simple_xmlproc.c 中的例程的客户机程序。
	simple_xmlproc_create.db2	要注册 simple_xmlproc.c 中的存储过程的 CLP 脚本。
	simple_xmlproc_drop.db2	要删除 simple_xmlproc.c 中的存储过程的 CLP 脚本。
C	xmlinsert.sqc	将 XML 文档插入到 XML 数据类型的列中。
	xmludfs.sqc	使用标量函数、有源函数、以 SQL 为主体的 UDF 和表 UDF 时，将 XML 数据类型作为输入参数来传递、声明 XML 数据类型局部变量并返回值。
	xmludfs.c	使用标量函数、有源函数、以 SQL 为主体的 UDF 和表 UDF 时，将 XML 数据类型作为输入参数来传递、声明 XML 数据类型局部变量并返回值。
	xmlupdel.sqc	更新和删除表中的 XML 文档。
	xmlread.sqc	读取存储在表中的 XML 数据。
	reltoxmltype.sqc	使用各种 SQL/XML 发布函数根据存储在关系表中的数据来创建 XML 对象。
	xmldecomposition.sqc	分解存储在 XML 文件中的数据并将该数据插入表中。指定在 XML 文档解析期间要使用的插入顺序。
	recxmldecomp.sqc	向 XSR 注册递归 XML 模式并启用它以进行分解。
	simple_xmlproc.sqc	具有 XML 类型参数的简单存储过程
	simple_xmlproc_client.db2	要调用 simple_xmlproc.sqc 中的例程的 CLP 脚本
	simple_xmlproc_create.db2	要注册 simple_xmlproc.sqc 中的存储过程的 CLP 脚本。
	simple_xmlproc_drop.db2	要删除 simple_xmlproc.sqc 中的存储过程的 CLP 脚本。
	xmltrig.sqc	使用触发器处理功能来强制自动验证入局 XML 文档。
	xmlintegrate.sqc	使用 XMLROW 和 XMLGROUP 函数将关系数据映射至 XML。说明 XMLQuery 缺省传递机制和 XMLTABLE 的缺省列规范。

表 47. pureXML － 应用程序开发样本 (续)

按语言归类的样本	样本程序名	程序描述
	xmlcheckconstraint.sqc	使用 IS VALIDATED 和 IS NOT VALIDATED 谓词创建在 XML 列上具有检查约束的表，并使用 ACCORDING TO XMLSCHEMA 子句指定一种或多种模式。
	xmlxslt.sqc	使用 XSLTRANSFORM 函数将数据库中的 XML 文档转换为 HTML、纯文本或其他使用样式表的 XML 格式。
CLP	xmlinsert.db2	将 XML 文档插入到 XML 数据类型的列中。
	xrpart.db2	在范围分区表中使用 XML 并且支持局部索引和全局索引。
	xmlpartition.db2	在分区数据库环境、MDC 和范围分区表中使用 XML。
	xmlmdc.db2	将数据从 MDC 表移至非 MDC 表，使用块索引和全局索引，以及更快插入和删除。
	xmludfs.db2	使用标量函数、有源函数、以 SQL 为主体的 UDF 和表 UDF 时，将 XML 数据类型作为输入参数来传递并返回值。
	xmldbafn.db2	使用内联 DBA 函数来确定 XML 文档的估计内联长度。
	xmlolic.db2	重新组织对表定义的索引以及如何重新组织范围分区表的非分区索引。
	xmlindgtt.db2	使用 XML 数据类型的已声明全局临时表。
	xmlupdel.db2	更新和删除表中的 XML 文档。
	reltoxmldoc.db2	使用各种 SQL/XML 发布函数直接根据存储在关系表中的数据来创建 XML 文档。
	reltoxmltype.db2	使用各种 SQL/XML 发布函数根据存储在关系表中的数据来创建 XML 对象。
	xmldecomposition.db2	分解存储在 XML 文件中的数据并将该数据插入表中。指定在 XML 文档解析期间要使用的插入顺序。
	recxmldecomp.db2	向 XSR 注册递归 XML 模式并启用它以进行分解。
	simple_xmlproc.db2	具有 XML 类型参数的简单存储过程

表 47. pureXML – 应用程序开发样本 (续)

按语言归类的样本	样本程序名	程序描述
	xmltotable.db2	使用 SQL/XML 函数（例如，XMLTABLE 和 XMLQUERY）和 XMLEXISTS 谓词将 XML 文档中的数据插入到关系表中。
	xmltrig.db2	使用触发器处理功能来强制自动验证入局 XML 文档。
	xmlintegrate.db2	使用 XMLROW 和 XMLGROUP 函数将关系数据映射至 XML。说明 XMLQuery 缺省传递机制和 XMLTABLE 的缺省列规范。
	xmlcheckconstraint.db2	使用 IS VALIDATED 和 IS NOT VALIDATED 谓词创建在 XML 列上具有检查约束的表，并使用 ACCORDING TO XMLSCHEMA 子句指定一种或多种模式。
	xmlxslt.db2	使用 XSLTRANSFORM 函数将数据库中的 XML 文档转换为 HTML、纯文本或其他使用样式表的 XML 格式。
JDBC	XmlInsert.java	将 XML 文档插入到 XML 数据类型的列中。
	XmlMdc.java	将数据从 MDC 表移至非 MDC 表，使用块索引和全局索引，以及更快插入和删除。
	XmlUdfs.java	使用标量函数、有源函数、以 SQL 为主体的 UDF 和表 UDF 时，将 XML 数据类型作为输入参数来传递并返回值。
	XmlUpDel.java	更新和删除表中的 XML 文档。
	XmlRead.java	读取存储在表中的 XML 数据。
	RelToXmlDoc.java	使用 SQL/XML 发布函数直接根据存储在关系表中的数据来创建 XML 文档。
	RelToXmlType.java	使用各种 SQL/XML 发布函数根据存储在关系表中的数据来创建 XML 对象。
	XmlDecomposition.java	分解存储在 XML 文件中的数据并将该数据插入表中。指定在 XML 文档解析期间要使用的插入顺序。
	RecXmlDecomp.java	向 XSR 注册递归 XML 模式并启用它以进行分解。
	Simple_XmlProc.java	具有 XML 类型参数的简单存储过程

表 47. pureXML – 应用程序开发样本 (续)

按语言归类的样本	样本程序名	程序描述
	Simple_XmlProc_Client.java	要调用 Simple_XmlProc.java 中的例程的客户机程序。
	Simple_XmlProc_Create.db2	要注册 Simple_XmlProc.java 中的存储过程的 CLP 脚本。
	Simple_XmlProc_Drop.db2	要删除 Simple_XmlProc.java 中的存储过程的 CLP 脚本。
	XmlToTable.java	使用 SQL/XML 函数（例如，XMLTABLE 和 XMLQUERY）和 XMLEXISTS 谓词将 XML 文档中的数据插入到关系表中。
	XmlTrig.java	使用触发器处理功能来强制自动验证入局 XML 文档。
	XmlCheckConstraint.java	使用 IS VALIDATED 和 IS NOT VALIDATED 谓词创建在 XML 列上具有检查约束的表，并使用 ACCORDING TO XMLSCHEMA 子句指定一种或多种模式。
SQLJ	XmlInsert.sqlj	将 XML 文档插入到 XML 数据类型的列中。
	XmlUpDel.sqlj	更新和删除表中的 XML 文档。
	XmlRead.sqlj	读取存储在表中的 XML 数据。
	RelToXmlDoc.sqlj	使用 SQL/XML 发布函数直接根据存储在关系表中的数据来创建 XML 文档。
	RelToXmlType.sqlj	使用 SQL/XML 发布函数根据存储在关系表中的数据来创建 XML 对象。
	XmlToTable.sqlj	使用 SQL/XML 函数（例如，XMLTABLE 和 XMLQUERY）和 XMLEXISTS 谓词将 XML 文档中的数据插入到关系表中。
	XmlIntegrate.sqlj	使用 XMLROW 和 XMLGROUP 函数将关系数据映射至 XML。说明 XMLQuery 缺省传递机制和 XMLTABLE 的缺省列规范。
	XmlXslt.sqlj	使用 XSLTRANSFORM 函数将数据库中的 XML 文档转换为 HTML、纯文本或其他使用样式表的 XML 格式。
PHP	XmlFlwor_DB2.php	使用 XQuery FLWOR 表达式。
	XmlIndex_DB2.php	创建一个索引并在 XQuery 中使用此索引。
	XmlInsert_DB2.php	将 XML 文档插入到 XML 数据类型的列中。
	XmlRead_DB2.php	读取存储在表中的 XML 数据。

表 47. pureXML – 应用程序开发样本 (续)

按语言归类的样本	样本程序名	程序描述
	XmlRelToXmlDOC_DB2.php	使用 SQL/XML 发布函数直接根据存储在关系表中的数据来创建 XML 文档。
	XmlRelToXmlType_DB2.php	使用 SQL/XML 发布函数根据关系数据和 XML 数据来创建 XML 文档。
	XmlRunstats_DB2.php	对包含 XML 类型列的表执行 RUNSTATS。
	XmlSchema_DB2.php	向数据库注册 XML 模式，然后使用已注册的模式来验证和插入 XML 文档。
	XmlSQLXQuery_DB2.php	使用 SQL/XML 查询。
	XmlUniqueIndexes_DB2.php	创建一个具有 UNIQUE 和 VARCHAR 长度约束的索引。
	XmlUpAndDel_DB2.php	更新和删除表中的 XML 文档。
	XmlToTable_DB2.php	使用 SQL/XML 将 XML 文档中的数据插入到关系表中。
	XmlXPath_DB2.php	运行简单的 XPath 查询。
	XmlXQuery_DB2.php	执行嵌套的 XQuery FLWOR 表达式。

第 11 章 XML 性能

下面的文章和准则包括使用 pureXML 功能部件时可遵循的性能调整注意事项。

pureXML 功能部件和数据组织方案

将 pureXML 功能部件与数据库分区环境、表分区以及多维集群之类的数据组织方案配合使用，可以显著提高查询性能，并降低数据维护操作（例如，重组、插入和删除）的开销。

CREATE TABLE 语句的下列三个子句包含用于指示应如何组织数据的算法:

- DISTRIBUTE BY，用于将数据平均分布在分区数据库环境中的数据库分区中（数据库分区）。
- PARTITION BY，用于指定表的表分区方案（表分区）。
- ORGANIZE BY，用于指定对表数据进行集群所使用的每个列或每组列的维（多维集群）。

请参阅"对 pureXML 功能部件的限制"以及"对基于 XML 的索引的限制"，以了解有关使用 pureXML 功能部件的限制的信息。

分区数据库环境

在分区数据库环境中使用表时，可以将包含 XML 列的表存储在多台机器上的一个多分区数据库中。可以使用 DB2 pureXML 功能部件来管理 XML 数据。

当 XML 数据分布在多个数据库分区中时，位于多台机器上的多个处理器可以处理对于信息的请求。数据检索和更新请求会被自动分解为子请求，并在适当的数据库分区中并行执行。

分区表

分区表可以包含一个或多个具有 XML 数据类型的列和基于 XML 数据的索引。用户创建的基于 XML 数据的索引只能是非分区索引。

多维集群表

多维集群（MDC）表可以包含具有 XML 数据类型的一列或多列，并且可以对 XML 列创建一个或多个基于 XML 数据的索引。可以将基于 XML 数据的索引与 MDC 块索引配合使用来提高查询性能。此外，可将 MDC 块索引与基于 XML 数据的索引配合使用来执行索引"与"（AND）运算。

在分区数据库环境中使用 XQuery 变换的示例

XQuery 变换表达式在分区数据库环境中受支持，并且可在使用抽取、变换和装入（ETL）操作的场景中使用。

例如，假定要连接从两个源表（ORDERS 和 PRODUCTS）中抽取的 XML 数据，将其变换为期望格式并作为 XML 数据插入到目标表 SALES 中。XQuery 变换表达式可用

于此用途，但它在表达式输入仅引用单个表而不是若干源表的连接时执行得最好。下面的示例说明了编写这类语句的一般技巧，此技巧确保可在多个分区间并行执行变换操作。

1. 在目标表所在的同一分区组中创建新的中间表，其分布键与目标表相同。该中间表可以是已声明全局临时表。
2. 从多个源表中抽取数据并将其插入到中间表中。
3. 变换中间表中的 XML 数据并将已变换数据插入到目标表中。

步骤 2 和步骤 3 可利用分区数据库环境中提供的并行处理以灵活执行。注意，在步骤 3 中应避免使用嵌套变换表达式。

示例

该示例使用源表 ORDERS 和 PRODUCTS、目标表 SALES 以及已声明全局临时表 TEMPSALES。目标是从 ORDER 表检索数据，将该数据与来自 PRODUCTS 表的订价信息相连接，然后格式化所产生的数据，并将格式化后的数据插入到 SALES 表中。

ORDERS 表包含每日订单及其标识（OID）和 XML 列（ORDERDETAIL）中的订单信息。每个 XML 文档包含客户标识（CID）和已订购产品。已订购产品信息包含产品标识、数量和送货要求。

```
CREATE TABLE ORDERS(OID BIGINT, ORDERDETAIL XML)DISTRIBUTE BY HASH (OID);
```

以下 INSERT 语句将样本订单插入到 ORDERS 表中:

```
INSERT into ORDERS
values (5003, '<order>
                <cid>1001</cid>
                <product>
                  <pid>2344</pid><qty>10</qty>
                  <delivery>Overnight</delivery>
                </product>
                <product>
                   <pid>537</pid><qty>3</qty>
                   <delivery>Ground</delivery>
                </product>
               </order>');
```

PRODUCTS 表包含产品信息、产品标识（PID）、产品价格（PRICE）以及 XML 列（PRODDETAIL）中有关产品的详细信息。

```
CREATE TABLE PRODUCTS(PID BIGINT, PRICE FLOAT, PRODDETAIL XML);
```

以下 INSERT 语句将两个产品插入到 PRODUCTS 表中:

```
INSERT into PRODUCTS
values(2344, 4.99, '<product>
                      <name>10 D-Cell batteries</name>
                      <desc>D Cell battery, 10-pack</desc>
                    </product>')

INSERT into PRODUCTS
values(537, 8.99, '<product>
                     <name>Ice Scraper, small</name>
                     <desc>Basic ice scraper, 4 inches wide</desc>
                   </product>');
```

SALES 表包含订单标识（OID）、客户标识（CID）、产品标识信息（PID）、订单总金额（ITEMTOTAL）以及 XML 列中存储的有关每个订单的详细信息（SALEDETAIL）。SALES 表的每行包含个别已订购产品的相关信息。SALES 表分布在订单标识（OID）列上。

```
CREATE TABLE SALES(OID BIGINT, CID BIGINT, PID BIGINT, ITEMTOTAL FLOAT,
    SALESDETAIL XML) DISTRIBUTE BY HASH (OID);
```

临时表与 SALE 表类似，用于存储中间查询结果。已声明全局临时表 TEMPSALES 用于处理每日订单。为了让 DB2 优化器正确优化用于更新临时表和 SALES 表的查询，临时表 TEMPSALES 和目标表 SALES 必须具有相同的分布键并属于同一分区组。

```
DECLARE GLOBAL TEMPORARY TABLE SESSION.TEMPSALES LIKE SALES
    DISTRIBUTE BY HASH (OID);
```

以下 SELECT 语句包含 PRODUCTS 与 ORDERS 表之间的连接。该语句使用 PRODUCTS 和 ORDERS 表中的信息来计算 ITEMTOTAL 列的值，并构造插入到 TEMPSALES 表中的 XML 文档。

```
INSERT INTO SESSION.TEMPSALES
SELECT O.OID, OX.CID, OX.PID, P.PRICE * OX.QTY, OX.SALESDETAIL
FROM PRODUCTS P,
     ORDERS O,
     XMLTABLE('for $i in $details/order/product
               return document{<info> {$details/order/cid} {$i} </info>}'
               passing O.ORDERDETAIL as "details"
     columns
        CID bigint path './info/cid',
        PID bigint path './info/product/pid',
        QTY int    path './info/product/qty',
        SALESDETAIL xml path '.') as OX
WHERE P.PID = OX.PID;
```

INSERT 语句使用的上一个 SELECT 语句未包含 XQuery 变换表达式。

临时表包含 SALES 表所需的关系信息。在临时表内的 XML 文档中，需要除去产品信息并且需要添加订单日期。

以下 INSERT 语句从临时表中选择每日销售信息并将该信息插入到表 SALES 中。INSERT 语句包含使用 XQuery 变换表达式的 SELECT 语句。在将每个 XML 文档插入到 SALE 表中之前，该表达式会从临时表修改该 XML 文档。因为 TEMPSALES 和 SALES 表在同一分区组中，并且共享一个公共分布键，所以可并行执行插入。

```
INSERT into SALES
select T.OID, T.CID, T.PID, T.ITEMTOTAL,
   XMLQUERY('transform
     copy $new := $temp
     modify (do delete ($new/info/cid,
     $new/info/product/pid,
     $new/info/product/qty),
       do insert <orderdate>{fn:current-date()}</orderdate>
       as first into $new/info)
    return $new' passing T.SALESDETAIL as "temp")
from SESSION.TEMPSALES T;
```

将已声明临时表与 XML 数据配合使用

实现需要创建非持久表的解决方案时，已声明全局临时表可能很有用。例如，应用程序可创建已声明临时表来处理数据。应用程序的会话结束时，将废弃临时表。

已声明全局临时表仅在声明它的会话期间存在。该表不能与其他会话共享。会话结束时，将删除临时表的行以及描述。已声明临时表没有目录争用问题，原因是已声明临时表没有目录条目。

已声明全局临时表可包含 XML 列，并且 XML 数据可查询和更新。还可在分区数据库环境中使用已声明全局临时表。临时表包含指定为分区键的列时，在临时表中对 XML 数据执行的操作结果可分布在数据库分区中。

示例

以下示例假定带有公司职员信息的表包含 XML 数据。职员数据类似于以下职员信息:

```
<company name="MyFirstComany">
  <emp id="31201" salary="60000" gender="Female">
     <name>       <first>Laura</first>
      <last>Brown</last>
    </name>
    <dept id="M25">Finance</dept>
  </emp>
</company>
```

以下 CREATE 语句创建带有 XML 列的职员表。

```
CREATE TABLE COMPANYINFO (ID INT, DOC XML)
```

以下两个语句创建可与职员表配合使用的全局临时表。两个语句所创建临时表的列名和描述与 COMPANYINFO 表的列名和描述相同。第一个语句使用列定义创建临时表。第二个语句使用 LIKE 子句创建临时表。该表各列的名称和描述与与指定表各列的名称和描述完全相同。ON COMMIT DELETE ROWS 子句指定，如果执行 COMMIT 操作时没有 WITH HOLD 游标对该表处于打开状态，那么会删除该表的所有行。

```
DECLARE GLOBAL TEMPORARY TABLE INSTMPTB (ID INT, DOC XML)
    ON COMMIT DELETE ROWS in USR_TBSP

DECLARE GLOBAL TEMPORARY TABLE INSTMPTB LIKE COMPANYINFO
    ON COMMIT DELETE ROWS in USR_TBSP
```

以下 DECLARE 语句创建其基本表行存储了 XML 文档的临时表。如果 XML 文档小于指定的直接插入长度，那么它将存储在基本表中。

```
DECLARE GLOBAL TEMPORARY TABLE TEMPTB (ID INT, DOC XML INLINE LENGTH 3000)
   ON COMMIT PRESERVE ROWS NOT LOGGED
```

如果要将大量数据插入到全局临时表中并对这些数据执行查询，那么可创建基于 XML 数据的索引以改进性能。例如，以下 CREATE INDEX 语句创建两个基于 XML 数据的索引。第一个索引基于 XML 文档中的职员标识。第二个索引基于职员的姓氏。

```
CREATE INDEX SESSION.TEMP_IDX ON SESSION.INSTMPTB (DOC)
  GENERATE KEY USING XMLPATTERN '/company/emp/@id'
  AS SQL DOUBLE
CREATE UNIQUE INDEX SESSION.TEMP_IDX2 ON SESSION.INSTMPTB (DOC)
  GENERATE KEY USING XMLPATTERN '/company/name/last'
  AS SQL VARCHAR(100)
```

可在分区数据库环境中创建已声明全局临时表以利用数据库分区。以下 DECLARE 语句通过将 DOCID 用作分布键来创建临时表。将 XML 数据添加至临时表之后，对 XML 数据的查询和其他操作可利用分区数据库环境。

```
DECLARE GLOBAL TEMPORARY TABLE INSTMPTB (ID INT, DOC XML)
   ON COMMIT DELETE ROWS
   IN USR_TBSP
   DISTRIBUTE BY HASH (ID)
```

以下 XQuery 表达式使用全局临时表和 COMPANYINFO 表。临时表包含 COMPANYINFO 表中的一部分文档。XQuery 表达式返回临时表中金融部门职员的 COMANYINFO 表的职员信息。

```
 XQUERY
   for $i in db2-fn:xmlcolumn("SESSION.INSTMPTB.DOC")/company/emp
   for $j in db2-fn:xmlcolumn("COMPANYINFO.DOC ")[/company/emp/@id = $i/@id ]
   where $i/dept = "Finance"
   return $j ;
```

以下 INSERT 语句将临时表中的数据插入到 COMPANYINFO 表中。

```
INSERT INTO COMPANYINFO FROM
 (SELECT ID, DOC FROM SESSION.INSTMPTB)
```

使用说明

使用已声明全局临时表时，以下各项适用:

- 已声明临时表支持数据行压缩。如果数据库管理器确定性能有所提升，那么会压缩表行数据，这些数据包括以直接插入方式存储在基本表对象中的 XML 文档。但是，不支持对已声明临时表的 XML 存储对象进行数据压缩。
- 在分区数据库环境中，使用 DISTRIBUTE BY 子句来创建带有分区键的已声明临时表。如果使用 LIKE 子句创建已声明临时表并且源表具有分区键，那么该临时表将没有分区键。
- 您对已声明临时表创建的索引支持索引压缩，这些已声明临时表包括用户创建的基于 XML 数据的索引。
- 用户创建的基于 XML 数据的索引与为非临时表创建的索引存在相同限制。例如，在分区数据库环境中，不支持唯一的基于 XML 数据的 XML 索引。

将优化准则与 XML 数据和 XQuery 表达式配合使用

优化概要文件中使用的 DB2 优化准则支持 XML 数据。可创建概要文件以使用准则来控制在分区数据库环境中移动 XML 数据的方法、XML 数据类型上的连接的顺序以及用户定义的基于 XML 数据的索引用法。

可在优化准则中对访问 XML 数据或使用基于 XML 数据的索引的查询指定下列类型的优化:

- 控制如何在分区数据库环境中的分区间使用 DPFXMLMOVEMENT 常规请求元素移动 XML 数据。
- 控制计划优化准则中连接 XML 数据类型的顺序，方法是在访问请求元素中将属性 FIRST 设置为 TRUE 或使用连接请求元素。
- 控制如何将基于 XML 数据的索引与访问请求元素配合使用:
  - 指定将单个 XML 索引扫描与 XISCAN 访问请求元素配合使用来访问表。
  - 指定将多个进行了 XANDOR 运算的 XML 索引扫描与 XANDOR 访问请求元素配合使用来访问表。

- 指定将多个关系索引和 XML 索引扫描与 IXAND 访问请求元素配合使用，并将 TYPE 属性值设置为 XMLINDEX。
- 指定让 DB2 优化器执行基于成本的分析并选择某个 XML 索引访问来访问表，方法是使用 ACCESS 访问请求元素并将属性 TYPE 设置为 XMLINDEX。例如，优化器可使用 XML 索引访问，如 XML 索引扫描、XANDOR 或带有至少一个 XML 索引的索引“与”（AND）运算。
- 指定使用所有适用的关系索引和基于 XML 数据的索引来访问指定表而不考虑成本，方法是使用 ACCESS 访问请求元素，并将属性 TYPE 设置为 XMLINDEX，将属性 ALLINDEXES 设置为 TRUE。
- 指定使用 INAND 计划中所有适用的关系索引和基于 XML 数据的索引来访问指定表而不考虑成本，方法是使用 IXAND 访问请求元素，并将属性 TYPE 设置为 XMLINDEX，将属性 ALLINDEXES 设置为 TRUE。

使用 XML 数据的优化准则的示例

示例优化概要文件包含常规请求、存取方法和连接顺序准则，用于控制如何对访问 XML 数据的查询执行优化。

移动作为引用的 XML 文档的准则

在以下优化概要文件中，准则中的 DPFXMLMOVEMENT 常规请求元素指定对 XML 文档的引用将通过访问方案中的 TQ 运算符进行移动。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables">
  <STMTKEY SCHEMA="ST">
    SELECT *
    FROM security
    WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
      /StockInfo[Industry= "OfficeSupplies"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <DPFXMLMOVEMENT VALUE="REFERENCE"/>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

使用基于 XML 数据的特定索引的准则

在以下优化概要文件中，准则中的 XISCAN 访问元素指定应使用索引 SEC_INDUSTRY 来访问表 SECURITY。如果 XISCAN 元素未使用 INDEX 属性指定索引名，那么优化器会使用基于 XML 数据的索引来访问表 SECURITY 以尽量降低成本。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables">
  <STMTKEY SCHEMA="ST">
    SELECT *
    FROM security
    WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
      /StockInfo[Industry= "OfficeSupplies"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <XISCAN TABLE='SECURITY' INDEX='SEC_INDUSTRY'/>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

将基于 XML 数据的索引与 XANDOR 访问配合使用的准则

在以下优化概要文件的准则中，XANDOR 元素指定应使用所有基于 XML 数据的适用索引的 XANDOR 计划来访问表 SECURITY。因为 XANDOR 计划不能使用关系索引，所以不会使用关系索引。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
    SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
           /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <XANDOR TABLE='SECURITY' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

将基于 XML 数据的多个指定索引与 IXAND 配合使用的准则

在以下优化概要文件中，优化准则中的 IXAND 元素指定应使用两个基于 XML 数据的索引（SEC_INDUSTRY 和 SEC_SYMBOL）来访问表 SECURITY。优化器会生成 INAND 计划，并按列示顺序将这两个索引作为 IXAND 计划的支柱。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
    SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
          /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <IXAND TABLE='SECURITY' TYPE='XMLINDEX'>
      <INDEX IXNAME='SEC_INDUSTRY'/>
      <INDEX IXNAME='SEC_SYMBOL'/>
    </IXAND>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

将所有基于 XML 数据的索引与 IXAND 配合使用的准则

在以下优化概要文件中，优化准则中的 IXAND 元素指定应使用所有适用的关系索引和基于 XML 数据的索引来访问表 SECURITY。假定 TRANS_DATE 有关系索引，SEC_INDUSTRY 和 SEC_SYMBOL 有基于 XML 数据的索引，那么所有三个索引将按优化器选择的顺序通过 AND 运算连接到一起。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
    SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
```

```
    AND XMLEXISTS('$SDOC/Security/SecurityInfo
         /StockInfo[Industry= "Software"]')
    AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <IXAND TABLE='SECURITY' TYPE='XMLINDEX' ALLINDEXES='TRUE' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

将基于 XML 数据的特定前导索引与 IXAND 配合使用的准则

在以下优化概要文件中，优化准则中的 IXAND 元素指定应使用 IXAND 计划访问表 SECURITY，并且基于 XML 数据的索引 SEC_INDUSTRY 必须是 IXAND 中的第一个索引。优化器应按基于成本的方式选择用于 IXAND 计划的其他索引。如果关系索引在列 TRANS_DATE 上可用，并且基于 XML 数据的索引在路径 SYMBOL 上可用，并且优化器认为有利，那么这两个索引中的一个或全部将作为 IXAND 计划的附加支柱出现。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
    SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
             /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <IXAND TABLE='SECURITY' TYPE='XMLINDEX' INDEX='SEC_INDUSTRY' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

使用某些 XML 索引访问的准则

在以下优化概要文件中，优化准则仅指定应使用某个基于 XML 数据的索引来访问表 SECURITY。优化器通过基于成本的分析来使用 XISCAN、IXAND、XANDOR 或 IXOR 计划。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables">
  <STMTKEY SCHEMA="ST">
     SELECT *
     FROM security
     WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
            /StockInfo[Industry= "OfficeSupplies"]')
  </STMTKEY>
  <OPTGUIDELINES>
     <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

使用所有基于 XML 数据的适用索引访问的准则

在以下优化概要文件中，优化准则指定使用 SECURITY 表的所有适用索引。方法由优化器进行选择。假定创建了以下两个基于 XML 数据的索引：SEC_INDUSTRY 和

SEC_SYMBOL，它们与 XMLEXISTS 中的两个谓词匹配。优化器选择使用 XANDOR 或 IXAND 计划。优化器使用基于成本的分析在两个访问方案之间进行选择。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables 2">
  <STMTKEY SCHEMA="ST2">
    SELECT *
    FROM security
    WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
              /StockInfo[Industry= "Software"]')
    AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' ALLINDEXES='TRUE' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

指定使用特定索引进行基于 XML 数据的索引的访问的准则

在以下优化概要文件中，准则指定使用至少一个 SEC_INDUSTRY 索引来访问表 SECURITY。优化器使用基于成本的分析来选择下列其中一个访问方案:

1. 使用 SEC_INDUSTRY 索引的 XISCAN 计划。
2. 将给定索引作为其第一个支柱的 IXAND 计划。优化器可能会根据基于成本的分析在 IXAND 计划中使用更多索引。在此示例中，如果关系索引在列 TRANS_DATE 上可用，并且优化器认为有利，那么该索引将作为 IXAND 计划的附加支柱出现。
3. 带有给定索引和基于 XML 数据的其他适用索引的 XANDOR 计划。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
    SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
              /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' INDEX='SEC_INDUSTRY' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

以下优化准则指定通过基于成本的分析使用下列其中一个访问方案:

- 将 IXAND 与 SEC_INDUSTRY 和 SEC_SYMBOL 索引（按指定顺序）配合使用。
- 将 XANDOR 计划与所有适用 XML 索引配合使用。

```
<OPTGUIDELINES>
  <ACCESS TABLE='SECURITY' TYPE='XMLINDEX'>
    <INDEX IXNAME='SEC_INDUSTRY'/>
    <INDEX IXNAME='SEC_SYMBOL'/>
  </ACCESS>
</OPTGUIDELINES>
```

控制连接顺序和指定基于 XML 数据的索引访问的准则

在以下优化概要文件中，优化准则包含两个元素。第一个准则元素指定，连接 FROM 子句中的表时表 CUSTACC 必须作为最外部表出现，并且必须使用某个基于 XML 数据

的索引访问来访问表 CUSTACC。第二个准则元素指定应使用 XANDOR 计划来访问表 ORDER。优化器在 XANDOR 计划中使用所有基于 XML 数据的适用索引。索引顺序由优化器进行选择。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Order and Security Tables">
  <STMTKEY SCHEMA="OST">
     SELECT ordqty, orddate, ordid, security, lasttrade
     FROM order, security, custacc,
       XMLTABLE('$ODOC/FIXML/Order'
          COLUMNS ordid VARCHAR(10) PATH '@ID',
                  orddate date PATH '@TrdDt',
                  ordqty float PATH 'OrdQty/@Qty') AS T1,
       XMLTABLE(' $SDOC/Security'
          COLUMNS security varchar(50) PATH 'Name',
                  lasttrade float PATH 'Price/LastTrade') AS T2
          WHERE XMLEXISTS('
         $SDOC/Security[Symbol/fn:string(.)
             = $ODOC/FIXML/Order/Instrmt/@Sym/fn:string(.)]')
       and XMLEXISTS(
         '$ODOC/FIXML/Order[@Acct/fn:string(.)
              = $CADOC/Customer/Accounts/Account/@id/fn:string(.)]')
        and XMLEXISTS('$CADOC/Customer[@id = 1011]')
       ORDER BY ordqty desc
  </STMTKEY>
  <OPTGUIDELINES>
      <ACCESS TABLE='CUSTACC' TYPE='XMLINDEX' FIRST='TRUE' />
      <XANDOR TABLE='ORDER' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

XQuery 表达式的准则

以下优化概要文件包含 XQuery 表达式，该表达式返回表 SECURITY1 中经营软件的公司的库存信息。该准则指定应使用单个 XML 索引 XI1 来访问该表。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Industry">
  <STMTKEY SCHEMA="STBD">
    xquery
    for $sinfo1 in db2-fn:xmlcolumn("SECURITY1.SDOC")/Security/SecurityInfo
        /StockInfo[Industry="Software"]
    return $sinfo1
  </STMTKEY>
  <OPTGUIDELINES>
    <XISCAN TABLE='SECURITY1' INDEX='XI1'/>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

以下优化概要文件包含 XQuery 表达式，该表达式返回经营软件和电子设备的公司的库存信息。

该准则指定表 SECURITY2 应该是连接中的外部表，并且应该使用基于 XML 数据的索引 XI2 来访问表 SECURITY2。该准则还会指定表 SECURITY1 应该是连接的内部表，并且应该使用基于 XML 数据的索引 XI1 来访问 SECURITY1 表。

```
<?xml version="1.0" encoding="UTF-8">
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Industry">
  <STMTKEY SCHEMA="STBD">
```

```
    <![CDATA[ xquery
     for $sinfo1 in db2-fn:xmlcolumn("SECURITY1.SDOC")/Security
          /SecurityInfo/StockInfo[Industry="Software"]
     for $sinfo2 in db2-fn:xmlcolumn("SECURITY2.SDOC")/Security
          /SecurityInfo/StockInfo[Industry="Electronics"]
     where $sinfo1 = $sinfo2
     return <stock> {$sinfo1} </stock> ]]>
  </STMTKEY>
  <OPTGUIDELINES>
     <JOIN>
        <ACCESS TABLE='SECURITY2' TYPE='XMLINDEX' INDEX='XI2'/>
        <ACCESS TABLE='SECURITY1' TYPE='XMLINDEX' INDEX='XI1'/>
     </JOIN>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

CDATA 部分以 <![CDATA[and ending with]]> 开头，并在 STMTKEY 元素中包括语句关键字，原因是该语句关键字包含特殊 XML 字符 < 和 >。概要文件解析器忽略 CDATA 部分中的 XML 标记，但优化器仍将使用整个语句关键字来使语句概要文件与应用程序中的对应语句相匹配。

用于 pureXML 数据存储性能的 DMS 表空间的首选项

对于性能敏感的应用程序，特别是涉及大量 INSERT 活动的应用程序，强烈建议您使用数据库管理空间（DMS）的表空间。

如果使用 pureXML 数据存储时发现查询性能下降并且正在使用系统管理空间（SMS），那么应该考虑切换至 DMS。

使用 DMS 功能还允许您利用 DB2 中的自主功能。

第 12 章 XML 数据编码

XML 数据的编码可以从数据本身派生（称为*内部编码*数据），也可以从外部源派生（称为*外部编码*数据）。

用来在应用程序与 XML 列之间交换 XML 数据的应用程序数据类型确定了编码的派生方式。

- 将字符或图形应用程序数据类型的 XML 数据视为外部编码的数据。与字符和图形数据相同，将这些数据类型的 XML 数据视为使用应用程序代码页编码的数据。
- 将二进制应用程序数据类型的 XML 数据或者字符数据类型的二进制数据视为内部编码的数据。

外部编码的 XML 数据可以包含内部编码，例如，当字符数据类型的 XML 文档包含编码声明时，情况即如此。将外部编码的数据发送至 DB2 数据库时，数据库管理器将检查内部编码。

如果外部编码和内部编码不是 Unicode 编码，那么与内部编码关联的有效 CCSID 必须与外部编码匹配。否则，将发生错误。如果外部编码和内部编码是 Unicode 编码，但编码方案不匹配，那么 DB2 数据库服务器将忽略内部编码。

内部编码的 XML 数据

二进制应用程序数据类型的 XML 数据具有内部编码。这样，数据的内容就确定了编码。DB2 数据库系统按照 XML 标准来根据文档内容派生内部编码。

内部编码是根据三个组件派生的:

Unicode 字节顺序标记（BOM）

由 XML 数据开头的 Unicode 字符代码组成的字节序列。BOM 指示后续文本的字节顺序。DB2 数据库管理器仅识别 XML 数据的 BOM。对于存储在非 XML 列中的 XML 数据来说，数据库管理器将 BOM 值视为与任何其他字符或二进制值相同。

XML 声明

XML 文档开头的处理指令。此声明提供有关其余 XML 内容的特定详细信息。

编码声明

XML 声明的可选部分，此声明指定文档中字符的编码。

DB2 数据库管理器使用以下过程来确定编码:

1. 如果数据包含 Unicode BOM，那么编码由 BOM 确定。下表列示了 BOM 类型以及产生的数据编码:

表 48. 字节顺序标记以及产生的文档编码

BOM 类型	BOM 值	编码
UTF-8	X’EFBBBF’	UTF-8
UTF-16 大尾数法	X’FEFF’	UTF-16
UTF-16 小尾数法	X’FFFE’	UTF-16

表 48. 字节顺序标记以及产生的文档编码 (续)

BOM 类型	BOM 值	编码
UTF-32 大尾数法	X'0000FEFF'	UTF-32
UTF-32 小尾数法	X'FFFE0000'	UTF-32

2. 如果数据包含 XML 声明，那么编码取决于是否有编码声明:
   - 如果有编码声明，那么编码是 encoding 属性值。例如，对于带有以下 XML 声明的 XML 数据来说，编码是 EUC-JP:

     ```
     <?xml version="1.0" encoding="EUC-JP"?>
     ```
   - 如果有编码声明和 BOM，那么编码声明必须与 BOM 中的编码匹配。否则，将发生错误。
   - 如果既没有编码声明也没有 BOM，那么数据库管理器根据 XML 声明的编码来确定编码:
     - 如果 XML 声明使用单字节 ASCII 字符，那么文档编码是 UTF-8。
     - 如果 XML 声明使用双字节 ASCII 字符，那么文档编码是 UTF-16。
3. 如果既没有 XML 声明也没有 BOM，那么文档编码是 UTF-8。

存储或传递 XML 数据时的编码注意事项

XML 数据必须正确进行编码才能存储在 DB2 表中。当从表中检索到数据并与 DB2 存储过程或者用户定义的函数配合使用时，或者与外部 Java 应用程序配合使用时，必须考虑编码。

将 XML 数据输入数据库时的编码注意事项

在 DB2 表中存储 XML 数据时，必须考虑内部和外部编码。

需要遵守以下规则:

- 如果内部编码和外部编码不是 Unicode 编码，那么对于外部编码的 XML 数据（使用字符数据类型发送至数据库服务器的数据）来说，任何内部编码的声明*必须*与外部编码匹配。否则，将发生错误，并且数据库管理器将拒绝该文档。

  如果外部编码和内部编码是 Unicode 编码，但编码方案不匹配，那么 DB2 数据库服务器将忽略内部编码。
- 对于内部编码的 XML 数据（使用二进制数据类型发送至数据库服务器的数据）来说，应用程序*必须*确保数据包含准确的编码信息。

从数据库中检索 XML 数据时的编码注意事项

从 DB2 表中检索 XML 数据时，需要避免数据丢失和截断。当无法使用目标数据的编码来表示源数据字符时，就会丢失数据。在转换到目标数据类型时，如果会导致扩展数据，就可能会发生数据截断。

由于 Java 和 .NET 字符串数据类型使用 Unicode UTF-16 或 UCS-2 编码，所以，Java 和 .NET 应用程序并不会像其他类型应用程序那样容易发生数据丢失问题。将 UTF-8 字符转换为 UTF-16 或 UCS-2 编码时，由于会发生扩展，所以有可能发生截断。

在例程参数中传递 XML 数据时的编码注意事项

在 DB2 数据库系统中，许多 XML 数据类型可用于存储过程或用户定义的函数定义中的参数。

以下 XML 数据类型可用:

XML 用于 SQL 过程。

XML AS CLOB
用于外部 SQL 过程和外部用户定义的函数。

如果应用程序编码不是 UTF-8，那么会对 XML AS CLOB 参数中的数据进行字符转换。在外部用户定义的函数或存储过程中，应该避免字符转换开销。在调用应用程序中，参数可以使用任何应用程序字符数据类型或图形数据类型，但源数据不应包含编码声明。可能会执行其他代码页转换操作，这会导致编码信息不准确。如果在应用程序中对该数据进行进一步解析，就可能会导致数据损坏。

JDBC、SQLJ 和 .NET 应用程序中的 XML 数据编码注意事项

通常，与 DB2 CLI 或嵌入式 SQL 应用程序相比，Java 应用程序的 XML 编码注意事项较少。虽然内部编码的 XML 数据的编码注意事项对于所有应用程序来说都是相同的，但是，对于 Java 应用程序中外部编码的数据来说，由于应用程序代码页始终是 Unicode，所以情况得到简化。

在 Java 应用程序中输入 XML 数据时的一般建议

- 如果输入数据在文件中，请以二进制流（`setBinaryStream`）方式读取该数据，以便数据库管理器进程将其作为内部编码的数据来进行处理。
- 如果输入数据在 Java 应用程序变量中，那么您选择的应用程序变量类型决定 DB2 数据库管理器是否使用内部编码。如果将数据作为字符类型（例如 `setString`）输入，数据库管理器在存储该数据前将把该数据从 UTF-16（应用程序代码页）转换为 UTF-8。

在 Java 应用程序中输出 XML 数据时的一般建议

- 如果将 XML 数据作为非二进制数据输出到文件中，那么应该对输出数据添加 XML 内部编码。

  文件系统的编码可能不是 Unicode，因此，在将字符串数据存储到文件时可能会对其执行转换。如果将数据作为二进制数据写入文件，那么不会执行转换。

  对于 Java 应用程序来说，数据库服务器不会为隐式的 XML 序列化操作添加显式的声明。如果对输出数据进行类型转换以将其转换为 `com.ibm.db2.jcc.DB2Xml`，并且调用其中一个 `getDB2Xml`*xxx* 方法，JDBC 驱动程序就会添加编码声明，如下表所示。

getDB2Xml*xxx*	**声明中的编码**
getDB2XmlString	ISO-10646-UCS-2
getDB2XmlBytes(String *targetEncoding*)	*targetEncoding* 指定的编码
getDB2XmlAsciiStream	US-ASCII
getDB2XmlCharacterStream	ISO-10646-UCS-2
getDB2XmlBinaryStream(String *targetEncoding*)	*targetEncoding* 指定的编码

对于指定了 INCLUDING XMLDECLARATION 的显式 XMLSERIALIZE 函数来说，数据库服务器将添加编码，并且 JDBC 驱动程序不会修改该编码。数据库服务器添加的显式编码是 UTF-8 编码。根据应用程序检索值时采用方式的不同，数据的实际编码可能与显式的内部编码不匹配。

- 如果应用程序将输出数据发送至 XML 解析器，那么应该使用 UTF-8、UCS-2 或 UTF-16 编码来在二进制应用程序变量中检索该数据。

XML 编码和序列化对于数据转换的影响

在数据库与应用程序之间传递数据时，XML 序列化的方法以及在内部或外部用于指定 XML 数据编码的方法将影响 XML 数据的转换。

编码情况：将内部编码的 XML 数据输入到数据库中

示例说明在将 XML 数据输入 XML 列时内部编码对数据转换和截断的影响。

通常，使用二进制应用程序数据类型能够最大程度地减少将数据输入数据库期间的代码页转换问题。

情况 1

编码源	值
数据编码	UTF-8 Unicode 输入数据，带有或不带 UTF-8 BOM 或 XML 编码声明
应用程序数据类型	二进制
应用程序代码页	不适用

输入语句示例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
   (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

字符转换: 不进行字符转换。

丢失数据: 不会丢失数据。

截断: 不会截断数据。

情况 2

编码源	值
数据编码	UTF-16 Unicode 输入数据，包含 UTF-16 BOM 或 XML 编码声明
应用程序数据类型	二进制
应用程序代码页	不适用

输入语句示例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
   (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

字符转换: DB2 数据库服务器执行 XML 解析时将把数据从 UTF-16 转换为 UTF-8 以便存储在 XML 列中。

丢失数据: 不会丢失数据。

截断: 不会截断数据。

情况 3

编码源	值
数据编码	ISO-8859-1 输入数据，包含 XML 编码声明
应用程序数据类型	二进制
应用程序代码页	不适用

输入语句示例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
    (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

字符转换: DB2 数据库系统执行 XML 解析时将把数据从 CCSID 819 转换为 UTF-8 以便存储在 XML 列中。

丢失数据: 不会丢失数据。

截断: 不会截断数据。

情况 4

编码源	值
数据编码	Shift_JIS 输入数据，包含 XML 编码声明
应用程序数据类型	二进制
应用程序代码页	不适用

输入语句示例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
    (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

字符转换: DB2 数据库系统执行 XML 解析时将把数据从 CCSID 943 转换为 UTF-8 以便存储在 XML 列中。

丢失数据: 不会丢失数据。

截断: 不会截断数据。

编码情况: 将外部编码的 XML 数据输入到数据库中

示例说明在将 XML 数据输入 XML 列时外部编码对数据转换和截断的影响。

通常，使用字符应用程序数据类型时，在将数据输入数据库期间不存在代码页转换问题。

由于 Java 和 .NET 应用程序的应用程序代码页始终是 Unicode，所以只有情况 1 和请求 2 适用于 Java 和 .NET 应用程序。

情况 1

编码源	值
数据编码	UTF-8 Unicode 输入数据，带有或不带适当的编码声明或 BOM
应用程序数据类型	字符
应用程序代码页	1208（UTF-8）

输入语句示例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
   (XMLPARSE(DOCUMENT CAST(? AS CLOB) PRESERVE WHITESPACE))
```

字符转换: 不进行字符转换。

丢失数据: 不会丢失数据。

截断: 不会截断数据。

情况 2

编码源	值
数据编码	UTF-16 Unicode 输入数据，带有或不带适当的编码声明或 BOM
应用程序数据类型	图形
应用程序代码页	任何 SBCS 代码页或 CCSID 1208

输入语句示例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
   (XMLPARSE(DOCUMENT CAST(? AS DBCLOB) PRESERVE WHITESPACE))
```

字符转换: DB2 数据库系统执行 XML 解析时将把数据从 UTF-16 转换为 UTF-8 以便存储在 XML 列中。

丢失数据: 不会丢失数据。

截断: 由于会进行扩展，所以在从 UTF-16 转换为 UTF-8 期间可能会发生截断。

情况 3

编码源	值
数据编码	ISO-8859-1 输入数据，带有或不带适当的编码声明
应用程序数据类型	字符

编码源	值
应用程序代码页	819

输入语句示例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
    (XMLPARSE(DOCUMENT CAST(? AS CLOB) PRESERVE WHITESPACE))
```

字符转换: DB2 数据库系统执行 XML 解析时将把数据从 CCSID 819 转换为 UTF-8 以便存储在 XML 列中。

丢失数据: 不会丢失数据。

截断: 不会截断数据。

情况 4

编码源	值
数据编码	Shift_JIS 输入数据，带有或不带适当的编码声明
应用程序数据类型	图形
应用程序代码页	943

输入语句示例:

```
INSERT INTO T1 VALUES (?)
INSERT INTO T1 VALUES
    (XMLPARSE(DOCUMENT CAST(? AS DBCLOB)))
```

字符转换: DB2 数据库系统执行 XML 解析时将把数据从 CCSID 943 转换为 UTF-8 以便存储在 XML 列中。

丢失数据: 不会丢失数据。

截断: 不会截断数据。

编码情况：通过隐式的序列化操作来检索 XML 数据

示例说明通过隐式的序列化操作检索 XML 数据时，目标编码和应用程序代码页对数据转换、截断和内部编码的影响。

由于 Java 应用程序的应用程序代码页始终是 Unicode，所以只有情况 1 和请求 2 适用于 Java 和 .NET 应用程序。通常，对于 Java 和 .NET 应用程序来说不存在代码页转换问题。

情况 1

编码源	值
目标数据编码	UTF-8 Unicode
目标应用程序数据类型	二进制

编码源	值
应用程序代码页	不适用

输出语句示例:

```
SELECT XMLCOL FROM T1
```

字符转换: 不进行字符转换。

丢失数据: 不会丢失数据。

截断: 不会截断数据。

序列化数据中的内部编码: 对于除 Java 或 .NET 应用程序以外的应用程序来说，数据使用以下 XML 声明作为前缀:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

对于 Java 或 .NET 应用程序，除非将数据转换为 `com.ibm.db2.jcc.DB2Xml` 类型并使用 `getDB2Xml`*xxx* 方法检索数据，否则不会添加编码声明。添加的声明取决于使用的 `getDB2Xml`*xxx*。

情况 2

编码源	值
目标数据编码	UTF-16 Unicode
目标应用程序数据类型	图形
应用程序代码页	任何 SBCS 代码页或 CCSID 1208

输出语句示例:

```
SELECT XMLCOL FROM T1
```

字符转换: 将数据从 UTF-8 转换为 UTF-16。

丢失数据: 不会丢失数据。

截断: 由于会进行扩展，所以在从 UTF-8 转换为 UTF-16 期间可能会发生截断。

序列化数据中的内部编码: 对于除 Java 或 .NET 应用程序以外的应用程序来说，数据使用 UTF-16 字节顺序标记（BOM）和以下 XML 声明作为前缀:

```
<?xml version="1.0" encoding="UTF-16" ?>
```

对于 Java 或 .NET 应用程序，除非将数据转换为 `com.ibm.db2.jcc.DB2Xml` 类型并使用 `getDB2Xml`*xxx* 方法检索数据，否则不会添加编码声明。添加的声明取决于使用的 `getDB2Xml`*xxx*。

情况 3

编码源	值
目标数据编码	ISO-8859-1 数据

编码源	值
目标应用程序数据类型	字符
应用程序代码页	819

输出语句示例:

```
SELECT XMLCOL FROM T1
```

字符转换: 将数据从 UTF-8 转换为 CCSID 819。

丢失数据: 可能会丢失数据。某些 UTF-8 字符在 CCSID 819 中无法表示。DB2 数据库系统将生成错误。

截断: 不会截断数据。

序列化数据中的内部编码: 数据使用以下 XML 声明作为前缀:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

情况 4

编码源	值
目标数据编码	Windows-31J 数据（Shift_JIS 的超集）
目标应用程序数据类型	图形
应用程序代码页	943

输出语句示例:

```
SELECT XMLCOL FROM T1
```

字符转换: 将数据从 UTF-8 转换为 CCSID 943。

丢失数据: 可能会丢失数据。某些 UTF-8 字符在 CCSID 943 中无法表示。DB2 数据库系统将生成错误。

截断: 由于会进行扩展，所以在从 UTF-8 转换为 CCSID 943 期间可能会发生截断。

序列化数据中的内部编码: 数据使用以下 XML 声明作为前缀:

```
<?xml version="1.0" encoding="Windows-31J" ?>
```

编码情况: 使用显式的 XMLSERIALIZE 来检索 XML 数据

示例说明通过显式地调用 XMLSERIALIZE 来检索 XML 数据时，目标编码和应用程序代码页对数据转换、截断和内部编码的影响。

由于 Java 应用程序的应用程序代码页始终是 Unicode，所以只有情况 1 和请求 2 适用于 Java 和 .NET 应用程序。

情况 1

编码源	值
目标数据编码	UTF-8 Unicode
目标应用程序数据类型	二进制
应用程序代码页	不适用

输出语句示例:

```
SELECT XMLSERIALIZE(XMLCOL AS BLOB(1M) INCLUDING XMLDECLARATION) FROM T1
```

字符转换: 不进行字符转换。

丢失数据: 不会丢失数据。

截断: 不会截断数据。

序列化数据中的内部编码: 数据使用以下 XML 声明作为前缀:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

情况 2

编码源	值
目标数据编码	UTF-16 Unicode
目标应用程序数据类型	图形
应用程序代码页	任何 SBCS 代码页或 CCSID 1208

输出语句示例:

```
SELECT XMLSERIALIZE(XMLCOL AS CLOB(1M) EXCLUDING XMLDECLARATION) FROM T1
```

字符转换: 将数据从 UTF-8 转换为 UTF-16。

丢失数据: 不会丢失数据。

截断: 由于会进行扩展，所以在从 UTF-8 转换为 UTF-16 期间可能会发生截断。

序列化数据中的内部编码: 由于指定了 EXCLUDING XMLDECLARATION，所以没有内部编码。如果指定了 INCLUDING XMLDECLARATION，那么内部编码指示 UTF-8 而不是 UTF-16。这会导致应用程序进程无法解析的 XML 数据依赖于编码名。

情况 3

编码源	值
目标数据编码	ISO-8859-1 数据
目标应用程序数据类型	字符
应用程序代码页	819

输出语句示例:

```
SELECT XMLSERIALIZE(XMLCOL AS CLOB(1M) EXCLUDING XMLDECLARATION) FROM T1
```

字符转换: 将数据从 UTF-8 转换为 CCSID 819。

丢失数据: 可能会丢失数据。某些 UTF-8 字符在 CCSID 819 中无法表示。如果某个字符在 CCSID 819 中无法表示，DB2 数据库管理器就会在输出中插入替换字符并发出警告。

截断: 不会截断数据。

序列化数据中的内部编码: 由于指定了 EXCLUDING XMLDECLARATION，所以没有内部编码。如果指定了 INCLUDING XMLDECLARATION，数据库管理器就会为 UTF-8 而不是 ISO-8859-1 添加内部编码。这会导致应用程序进程无法解析的 XML 数据依赖于编码名。

情况 4

编码源	值
目标数据编码	Windows-31J 数据（Shift_JIS 的超集）
目标应用程序数据类型	图形
应用程序代码页	943

输出语句示例:

```
SELECT XMLSERIALIZE(XMLCOL AS CLOB(1M) EXCLUDING XMLDECLARATION) FROM T1
```

字符转换: 将数据从 UTF-8 转换为 CCSID 943。

丢失数据: 可能会丢失数据。某些 UTF-8 字符在 CCSID 943 中无法表示。如果某个字符在 CCSID 943 中无法表示，数据库管理器就会在输出中插入替换字符并发出警告。

截断: 由于会进行扩展，所以在从 UTF-8 转换为 CCSID 943 期间可能会发生截断。

序列化数据中的内部编码: 由于指定了 EXCLUDING XMLDECLARATION，所以没有内部编码。如果指定了 INCLUDING XMLDECLARATION，那么内部编码指示 UTF-8 而不是 Windows-31J。这会导致应用程序进程无法解析的 XML 数据依赖于编码名。

映射内部编码的 XML 数据和 CCSID

在将数据从 XML 数据转换为另一数据类型时，DB2 数据库管理器根据 XML 内部编码确定 CCSID，并在将数据转换为 XML 数据类型时根据 CCSID 确定 XML 内部编码名称。

将编码名映射至已存储的 XML 数据的有效 CCSID

如果 XML 列中存储的数据包含在二进制应用程序变量中，或者该数据是内部编码的 XML 类型，那么 DB2 数据库管理器就会检查该数据以确定编码。如果该数据包含编码声明，数据库管理器就会将编码名称映射至 CCSID。

表 49 列示了这些映射。如果编码名未包含在表 49 中，数据库管理器就会返回错误。

表 49 第一列中的规范化编码名是通过将编码名转换为大写并除去所有连字符、加号、下划线、冒号、句点和空格得到。例如，ISO88591 是 ISO 8859-1、ISO-8859-1 和 iso-8859-1 的规范化编码名。

表 49. 编码名和有效 CCSID

规范化编码名	CCSID
437	437
646	367
813	813
819	819
850	850
852	852
855	855
857	857
862	862
863	863
866	866
869	869
885913	901
885915	923
88591	819
88592	912
88595	915
88597	813
88598	62210
88599	920
904	904
912	912
915	915
916	916
920	920
923	923
ANSI1251	1251
ANSIX341968	367
ANSIX341986	367
ARABIC	1089
ASCII7	367
ASCII	367
ASMO708	1089
BIG5	950
CCSID00858	858

表 49. 编码名和有效 CCSID（续）

规范化编码名	CCSID
CCSID00924	924
CCSID01140	1140
CCSID01141	1141
CCSID01142	1142
CCSID01143	1143
CCSID01144	1144
CCSID01145	1145
CCSID01146	1146
CCSID01147	1147
CCSID01148	1148
CCSID01149	1149
CP00858	858
CP00924	924
CP01140	1140
CP01141	1141
CP01142	1142
CP01143	1143
CP01144	1144
CP01145	1145
CP01146	1146
CP01147	1147
CP01148	1148
CP01149	1149
CP037	37
CP1026	1026
CP1140	1140
CP1141	1141
CP1142	1142
CP1143	1143
CP1144	1144
CP1145	1145
CP1146	1146
CP1147	1147
CP1148	1148
CP1149	1149
CP1250	1250
CP1251	1251
CP1252	1252
CP1253	1253
CP1254	1254

表 49. 编码名和有效 CCSID（续）

规范化编码名	CCSID
CP1255	1255
CP1256	1256
CP1257	1257
CP1258	1258
CP1363	1363
CP1383	1383
CP1386	1386
CP273	273
CP277	277
CP278	278
CP280	280
CP284	284
CP285	285
CP297	297
CP33722	954
CP33722C	954
CP367	367
CP420	420
CP423	423
CP424	424
CP437	437
CP500	500
CP5346	5346
CP5347	5347
CP5348	5348
CP5349	5349
CP5350	5350
CP5353	5353
CP813	813
CP819	819
CP838	838
CP850	850
CP852	852
CP855	855
CP857	857
CP858	858
CP862	862
CP863	863
CP864	864
CP866	866

表 49. 编码名和有效 CCSID（续）

规范化编码名	CCSID
CP869	869
CP870	870
CP871	871
CP874	874
CP904	904
CP912	912
CP915	915
CP916	916
CP920	920
CP921	921
CP922	922
CP923	923
CP936	1386
CP943	943
CP943C	943
CP949	970
CP950	950
CP964	964
CP970	970
CPGR	869
CSASCII	367
CSBIG5	950
CSEBCDICCAFR	500
CSEBCDICDKNO	277
CSEBCDICES	284
CSEBCDICFISE	278
CSEBCDICFR	297
CSEBCDICIT	280
CSEBCDICPT	37
CSEBCDICUK	285
CSEBCDICUS	37
CSEUCKR	970
CSEUCPKDFMTJAPANESE	954
CSGB2312	1383
CSHPROMAN8	1051
CSIBM037	37
CSIBM1026	1026
CSIBM273	273
CSIBM277	277
CSIBM278	278

表 49. 编码名和有效 CCSID（续）

规范化编码名	CCSID
CSIBM280	280
CSIBM284	284
CSIBM285	285
CSIBM297	297
CSIBM420	420
CSIBM423	423
CSIBM424	424
CSIBM500	500
CSIBM855	855
CSIBM857	857
CSIBM863	863
CSIBM864	864
CSIBM866	866
CSIBM869	869
CSIBM870	870
CSIBM871	871
CSIBM904	904
CSIBMEBCDICATDE	273
CSIBMTHAI	838
CSISO128T101G2	920
CSISO146SERBIAN	915
CSISO147MACEDONIAN	915
CSISO2INTLREFVERSION	367
CSISO646BASIC1983	367
CSISO88596I	1089
CSISO88598I	916
CSISOLATIN0	923
CSISOLATIN1	819
CSISOLATIN2	912
CSISOLATIN5	920
CSISOLATIN9	923
CSISOLATINARABIC	1089
CSISOLATINCYRILLIC	915
CSISOLATINGREEK	813
CSISOLATINHEBREW	62210
CSKOI8R	878
CSKSC56011987	970
CSMACINTOSH	1275
CSMICROSOFTPUBLISHING	1004
CSPC850MULTILINGUAL	850

表 49. 编码名和有效 CCSID（续）

规范化编码名	CCSID
CSPC862LATINHEBREW	862
CSPC8CODEPAGE437	437
CSPCP852	852
CSSHIFTJIS	943
CSUCS4	1236
CSUNICODE11	1204
CSUNICODE	1204
CSUNICODEASCII	1204
CSUNICODELATIN1	1204
CSVISCII	1129
CSWINDOWS31J	943
CYRILLIC	915
DEFAULT	367
EBCDICATDE	273
EBCDICCAFR	500
EBCDICCPAR1	420
EBCDICCPBE	500
EBCDICCPCA	37
EBCDICCPCH	500
EBCDICCPDK	277
EBCDICCPES	284
EBCDICCPFI	278
EBCDICCPFR	297
EBCDICCPGB	285
EBCDICCPGR	423
EBCDICCPHE	424
EBCDICCPIS	871
EBCDICCPIT	280
EBCDICCPNL	37
EBCDICCPNO	277
EBCDICCPROECE	870
EBCDICCPSE	278
EBCDICCPUS	37
EBCDICCPWT	37
EBCDICCPYU	870
EBCDICDE273EURO	1141
EBCDICDK277EURO	1142
EBCDICDKNO	277
EBCDICES284EURO	1145
EBCDICES	284

表 49. 编码名和有效 CCSID（续）

规范化编码名	CCSID
EBCDICFI278EURO	1143
EBCDICFISE	278
EBCDICFR297EURO	1147
EBCDICFR	297
EBCDICGB285EURO	1146
EBCDICINTERNATIONAL500EURO	1148
EBCDICIS871EURO	1149
EBCDICIT280EURO	1144
EBCDICIT	280
EBCDICLATIN9EURO	924
EBCDICNO277EURO	1142
EBCDICPT	37
EBCDICSE278EURO	1143
EBCDICUK	285
EBCDICUS37EURO	1140
EBCDICUS	37
ECMA114	1089
ECMA118	813
ELOT928	813
EUCCN	1383
EUCJP	954
EUCKR	970
EUCTW	964
EXTENDEDUNIXCODEPACKEDFORMATFORJAPANESE	954
GB18030	1392
GB2312	1383
GBK	1386
GREEK8	813
GREEK	813
HEBREW	62210
HPROMAN8	1051
IBM00858	858
IBM00924	924
IBM01140	1140
IBM01141	1141
IBM01142	1142
IBM01143	1143
IBM01144	1144
IBM01145	1145
IBM01146	1146

表 49. 编码名和有效 CCSID (续)

规范化编码名	CCSID
IBM01147	1147
IBM01148	1148
IBM01149	1149
IBM01153	1153
IBM01155	1155
IBM01160	1160
IBM037	37
IBM1026	1026
IBM1043	1043
IBM1047	1047
IBM1252	1252
IBM273	273
IBM277	277
IBM278	278
IBM280	280
IBM284	284
IBM285	285
IBM297	297
IBM367	367
IBM420	420
IBM423	423
IBM424	424
IBM437	437
IBM500	500
IBM808	808
IBM813	813
IBM819	819
IBM850	850
IBM852	852
IBM855	855
IBM857	857
IBM862	862
IBM863	863
IBM864	864
IBM866	866
IBM867	867
IBM869	869
IBM870	870
IBM871	871
IBM872	872

表 49. 编码名和有效 CCSID（续）

规范化编码名	CCSID
IBM902	902
IBM904	904
IBM912	912
IBM915	915
IBM916	916
IBM920	920
IBM921	921
IBM922	922
IBM923	923
IBMTHAI	838
IRV	367
ISO10646	1204
ISO10646UCS2	1200
ISO10646UCS4	1232
ISO10646UCSBASIC	1204
ISO10646UNICODELATIN1	1204
ISO646BASIC1983	367
ISO646IRV1983	367
ISO646IRV1991	367
ISO646US	367
ISO885911987	819
ISO885913	901
ISO885915	923
ISO885915FDIS	923
ISO88591	819
ISO885921987	912
ISO88592	912
ISO885951988	915
ISO88595	915
ISO885961987	1089
ISO88596	1089
ISO88596I	1089
ISO885971987	813
ISO88597	813
ISO885981988	62210
ISO88598	62210
ISO88598I	916
ISO885991989	920
ISO88599	920
ISOIR100	819

表 49. 编码名和有效 CCSID (续)

规范化编码名	CCSID
ISOIR101	912
ISOIR126	813
ISOIR127	1089
ISOIR128	920
ISOIR138	62210
ISOIR144	915
ISOIR146	915
ISOIR147	915
ISOIR148	920
ISOIR149	970
ISOIR2	367
ISOIR6	367
JUSIB1003MAC	915
JUSIB1003SERB	915
KOI8	878
KOI8R	878
KOI8U	1168
KOREAN	970
KSC56011987	970
KSC56011989	970
KSC5601	970
L1	819
L2	912
L5	920
L9	923
LATIN0	923
LATIN1	819
LATIN2	912
LATIN5	920
LATIN9	923
MAC	1275
MACEDONIAN	915
MACINTOSH	1275
MICROSOFTPUBLISHING	1004
MS1386	1386
MS932	943
MS936	1386
MS949	970
MSKANJI	943
PCMULTILINGUAL850EURO	858

表 49. 编码名和有效 CCSID（续）

规范化编码名	CCSID
R8	1051
REF	367
ROMAN8	1051
SERBIAN	915
SHIFTJIS	943
SJIS	943
SUNEUGREEK	813
T101G2	920
TIS20	874
TIS620	874
UNICODE11	1204
UNICODE11UTF8	1208
UNICODEBIGUNMARKED	1200
UNICODELITTLEUNMARKED	1202
US	367
USASCII	367
UTF16	1204
UTF16BE	1200
UTF16LE	1202
UTF32	1236
UTF32BE	1232
UTF32LE	1234
UTF8	1208
VISCII	1129
WINDOWS1250	1250
WINDOWS1251	1251
WINDOWS1252	1252
WINDOWS1253	1253
WINDOWS1254	1254
WINDOWS1255	1255
WINDOWS1256	1256
WINDOWS1257	1257
WINDOWS1258	1258
WINDOWS28598	62210
WINDOWS31J	943
WINDOWS936	1386
XEUCTW	964
XMSWIN936	1386
XUTF16BE	1200
XUTF16LE	1202

表 49. 编码名和有效 CCSID（续）

规范化编码名	CCSID
XWINDOWS949	970

将 CCSID 映射至序列化 XML 输出数据的编码名

作为隐式或显式 XMLSERIALIZE 操作的一部分，DB2 数据库管理器可能会在序列化 XML 输出数据开头添加编码声明。

该声明的格式如下所示:

```
<?xml version="1.0" encoding="encoding-name"?>
```

通常，编码声明中的字符集标识描述输出字符串中的字符编码。例如，在将 XML 数据序列化为与目标应用程序数据类型对应的 CCSID 时，编码声明将描述目标应用程序变量 CCSID。应用程序执行指定了 INCLUDING XMLDECLARATION 的显式 XMLSERIALIZE 函数时的情况例外。如果指定了 INCLUDING XMLDECLARATION，数据库管理器就会为 UTF-8 生成编码声明。如果目标数据类型是 CLOB 或 DBCLOB 类型，就可能会执行其他代码页转换操作，这会导致编码信息不准确。如果在应用程序中对该数据进行进一步解析，就可能会导致数据损坏。

根据 XML 标准的规定，DB2 数据库管理器尽可能地选择 CCSID 的 IANA 注册表名。

表 50. CCSID 和对应的编码名

CCSID	编码名
37	IBM037
273	IBM273
277	IBM277
278	IBM278
280	IBM280
284	IBM284
285	IBM285
297	IBM297
367	US-ASCII
420	IBM420
423	IBM423
424	IBM424
437	IBM437
500	IBM500
808	IBM808
813	ISO-8859-7
819	ISO-8859-1
838	IBM-Thai
850	IBM850
852	IBM852
855	IBM855

表 50. CCSID 和对应的编码名 (续)

CCSID	编码名
857	IBM857
858	IBM00858
862	IBM862
863	IBM863
864	IBM864
866	IBM866
867	IBM867
869	IBM869
870	IBM870
871	IBM871
872	IBM872
874	TIS-620
878	KOI8-R
901	ISO-8859-13
902	IBM902
904	IBM904
912	ISO-8859-2
915	ISO-8859-5
916	ISO-8859-8-I
920	ISO-8859-9
921	IBM921
922	IBM922
923	ISO-8859-15
924	IBM00924
932	Shift_JIS
943	Windows-31J
949	EUC-KR
950	Big5
954	EUC-JP
964	EUC-TW
970	EUC-KR
1004	Microsoft-Publish
1026	IBM1026
1043	IBM1043
1047	IBM1047
1051	hp-roman8
1089	ISO-8859-6
1129	VISCII
1140	IBM01140
1141	IBM01141

表 50. CCSID 和对应的编码名 (续)

CCSID	编码名
1142	IBM01142
1143	IBM01143
1144	IBM01144
1145	IBM01145
1146	IBM01146
1147	IBM01147
1148	IBM01148
1149	IBM01149
1153	IBM01153
1155	IBM01155
1160	IBM-Thai
1161	TIS-620
1162	TIS-620
1163	VISCII
1168	KOI8-U
1200	UTF-16BE
1202	UTF-16LE
1204	UTF-16
1208	UTF-8
1232	UTF-32BE
1234	UTF-32LE
1236	UTF-32
1250	windows-1250
1251	windows-1251
1252	windows-1252
1253	windows-1253
1254	windows-1254
1255	windows-1255
1256	windows-1256
1257	windows-1257
1258	windows-1258
1275	MACINTOSH
1363	KSC_5601
1370	Big5
1381	GB2312
1383	GB2312
1386	GBK
1392	GB18030
4909	ISO-8859-7
5039	Shift_JIS

表 50. CCSID 和对应的编码名 (续)

CCSID	编码名
5346	windows-1250
5347	windows-1251
5348	windows-1252
5349	windows-1253
5350	windows-1254
5351	windows-1255
5352	windows-1256
5353	windows-1257
5354	windows-1258
5488	GB18030
8612	IBM420
8616	IBM424
9005	ISO-8859-7
12712	IBM424
13488	UTF-16BE
13490	UTF-16LE
16840	IBM420
17248	IBM864
17584	UTF-16BE
17586	UTF-16LE
62209	IBM862
62210	ISO-8859-8
62211	IBM424
62213	IBM862
62215	ISO-8859-8
62218	IBM864
62221	IBM862
62222	ISO-8859-8
62223	windows-1255
62224	IBM420
62225	IBM864
62227	ISO-8859-6
62228	windows-1256
62229	IBM424
62231	IBM862
62232	ISO-8859-8
62233	IBM420
62234	IBM420
62235	IBM424
62237	windows-1255

表 50. CCSID 和对应的编码名 (续)

CCSID	编码名
62238	ISO-8859-8-I
62239	windows-1255
62240	IBM424
62242	IBM862
62243	ISO-8859-8-I
62244	windows-1255
62245	IBM424
62250	IBM420

第 13 章 带注释的 XML 模式分解

带注释的 XML 模式分解有时称为“分解”或“分块”，它指的是将 XML 文档中的内容存储到关系表列中的过程。带注释的 XML 模式分解根据 XML 模式中指定的注释来进行分解。分解 XML 文档后，插入的数据会使用插入其中的列的 SQL 数据类型。

XML 模式由一个或多个 XML 模式文档组成。在带注释的 XML 模式分解（即基于模式的分解）中，通过使用分解注释对文档的 XML 模式添加注释，可以对分解进行控制。这些注释指定了诸如目标表名和用来存储 XML 数据的列、未标识目标表 SQL 模式时使用的缺省 SQL 模式、将 XML 数据插入到目标表中时使用的顺序，以及存储内容前要对其执行的变换之类的详细信息。请参阅分解注释总结以获取更多示例，那些示例说明了通过这些注释可以指定的内容。

带注释的模式文档必须存储在 XML 模式存储库（XSR）中并向该存储库注册。然后，必须对该模式启用分解。

在成功注册带注释的模式后，可以通过调用其中一个分解存储过程或执行 DECOMPOSE XML DOCUMENT 命令来执行单个 XML 文档的分解。要分解存储在某列中的多个 XML 文档，请使用 XDB_DECOMP_XML_FROM_QUERY 存储过程或 DECOMPOSE XML DOCUMENTS 命令。

请注意，可以禁用基于模式的分解，也可以使之失效。有关更多信息，请参阅有关禁用分解的主题。

带注释的 XML 模式分解的优点

带注释的 XML 模式分解可以是解决以下问题的解决方案：要将符合 XML 模式的 XML 文档存储在表中，但该模式与存储文档的表的定义不完全匹配。

在 XML 模式与表结构不明显匹配的情况下，可能需要调整 XML 模式和／或关系模式，以使文档适合表结构。但是，并非总是能够对 XML 或关系模式进行更改，或者这种更改可能非常昂贵，特别是在现有应用程序期望关系模式具有特定结构时。

带注释的 XML 模式分解允许您根据现有的或新的 XML 模式将文档分解成现有表或新表，从而解决了此问题。由于带注释的 XML 模式分解中提供的各种功能，所以可以实现上述操作。这些功能（它们表示为添加至 XML 模式文档的注释）使得可以灵活地将 XML 模式结构映射至关系表结构。

使用带注释的 XML 模式来分解 XML 文档

当您想要将 XML 文档的各个部分存储在一个或多个表的列中时，可以使用带注释的 XML 模式分解。此类分解功能根据已注册的带注释的 XML 模式中指定的注释来对 XML 文档进行分解，以便存储在表中。

要使用带注释的 XML 模式来分解 XML 文档：

1. 如果您要使用从较早版本的 DB2 数据库产品创建的数据库，那么请使用列表文件 xdb.lst 运行 BIND 命令，可在 sqllib/bnd 目录中找到该列表文件。

2. 使用 XML 分解注释来注释模式文档。
3. 注册模式文档并允许模式分解。
4. 如果任何属于 XML 模式的已注册模式文档已更改，那么必须再次注册此 XML 模式的所有文档，并且必须对该 XML 模式启用分解。
5. 根据要分解的 XML 文档所在的位置，使用所指示的其中一种方法并提供 XML 模式的 XSR 对象名:
   - 对于文件系统上的单个 XML 文档，使用下列其中一种方法:
     - 调用大小刚好适合于所分解文档大小的其中一个 XDBDECOMPXML 存储过程。[3]
     - 发出 DECOMPOSE XML DOCUMENT 命令。
   - 对于存储在二进制列或 XML 列中的一个或多个文档，使用下列其中一种方法:
     - 发出 DECOMPOSE XML DOCUMENTS 命令。
     - 调用 XDB_DECOMP_XML_FROM_QUERY 存储过程。

注册 XML 模式并对其启用分解

一旦成功地注册带注释的模式并对其启用分解，就可以使用它来分解 XML 文档。

先决条件

- 确保使用 XML 分解注释对 XML 模式中的至少一个元素或属性声明添加了注释。此带注释元素或属性必须是复杂类型的全局元素的子元素，或者本身是复杂类型的全局元素。
- 确保数据库中存在从带注释的模式文档集中引用的所有表和列，此模式文档集组成了 XML 模式。在该模式引用的每个表和对应于该模式的 XSR 对象之间创建依赖关系。
- 确保 **applheapsz** 配置参数至少设置为 1024。

过程

选择下列其中一种方法来注册 XML 模式并对其启用分解: [4]

- 存储过程:
  1. 调用 XSR_REGISTER 存储过程并传递主模式文档。
  2. 如果该 XML 模式由多个模式文档组成，请对每个尚未注册的模式文档调用 XSR_ADDSCHEMADOC 存储过程。
  3. 调用 XSR_COMPLETE 存储过程并将 **isusedfordecomposition** 参数设置为 1。
- 命令行:
  - 如果该 XML 模式仅由一个模式文档组成，请发出指定了 COMPLETE 和 ENABLE DECOMPOSITION 选项的 REGISTER XML SCHEMA 命令。
  - 如果该 XML 模式由多个模式文档组成:

---

3. 如果正在使用脚本或应用程序来分解若干个未知大小的文档，请考虑使用 DECOMPOSE XML DOCUMENT 命令来进行分解，而不要使用 XDBDECOMPXML 存储过程，这是因为该命令会自动调用适合于文档大小的存储过程。

4. 如果先前已使用以上任一方法注册了该 XML 模式，但尚未对其启用分解，那么可以通过发出指定了 ENABLE DECOMPOSITION 选项的 ALTER XSROBJECT SQL 语句来对该模式启用分解。

1. 对于除最后一个模式文档以外的每个模式文档，请发出 REGISTER XML SCHEMA 命令。
2. 对于最后一个尚未注册的模式文档来说，请发出指定了 COMPLETE 和 ENABLE DECOMPOSITION 选项的 REGISTER XML SCHEMA 命令。

- JDBC 接口：
  1. 调用 DB2Connection.registerDB2XMLSchema 方法并将 **isUsedForDecomposition** 布尔参数设置为 true 以启用分解。[5]

当对 XML 模式启用分解时，在该模式引用的每个表和对应于该模式的 XSR 对象之间创建依赖关系。这种依赖关系不允许对该模式中引用的任何表进行重命名。必须对 XML 模式的 XSR 对象禁用分解才能将引用的表重命名。可以在 SYSCAT.XSROBJECTDEP 目录视图中找到 XSR 对象引用的表。

多个 XML 文档分解示例

DECOMPOSE XML DOCUMENTS 命令使用单个 XML 模式来分解存储在二进制列或 XML 列中的一组 XML 文档。XML 文档数据根据模式中所指定的注释来存储在关系表的各列中。

示例

以下示例假定 ABC.SALESTAB 关系表中包含 SALESDOC 和 DOCID 这两列。SALESDOC 列包含 XML 文档，而 DOCID 列包含存储在 SALESDOC 中的 XML 文档的文档标识。所有文档与在 XML 模式存储库（XSR）中注册为 ABC.SALES 的 XML 模式对应，已通过分解信息对此模式添加了注释且可用于分解。调用以下 DECOMPOSE XML DOCUMENTS 命令，以使用 ABC.SALES 模式来分解存储在 ABC.SALESTAB.SALESDOC 中的所有文档：

```
DECOMPOSE XML DOCUMENTS IN 'SELECT DOCID, SALESDOC FROM ABC.SALESTAB'
  XMLSCHEMA ABC.SALES
  MESSAGES /home/myid/errors/errorreport.xml
```

或者，使用 XDB_DECOMP_XML_FROM_QUERY 存储过程来分解 XML 文档。以下存储过程将与前一命令执行相同的分解操作。

```
XDB_DECOMP_XML_FROM_QUERY ('ABC', 'SALES', 'SELECT DOCID, SALESDOC FROM SALESTAB',
  0, 0, 0, NULL, NULL, 1, numInput, numDecomposed,
  errorreportBuf);
```

以下命令将使用 CUST_SHRED 模式来分解 CUSTOMER 表的 INFO 中 CUSTID 大于 1003 的客户信息。此示例假定已在 XSR 中注册了 CUST_SHRED 模式。

```
DECOMPOSE XML DOCUMENTS IN 'SELECT CUSTID, INFO FROM CUSTOMER WHERE CUSTID > 1003'
  XMLSCHEMA CUST_SHRED
  MESSAGES /home/myid/errors/errorreport.xml
```

带注释的 XML 模式分解和递归 XML 文档

可在 XML 模式存储库（XSR）中注册包含递归的 XML 模式并对其启用分解，但存在以下限制：递归关系本身不能分解为目标表中的标量值。通过使用相应的模式注释，可通过序列化标记的方式存储并检索递归部分。

5. 此方法有两种格式：一种格式用于从 InputStream 对象输入的 XML 模式文档，另一种格式用于 String 中的 XML 模式文档。

递归类型

当 XML 模式中的类型定义允许具有相同名称和类型的元素出现在自己的定义中时，XML 模式会被认为是递归 XML 模式。递归可以是显式或隐式的。

显式递归

依据自身定义元素时，就发生显式递归。如以下示例中所示，通过使用 ref 元素声明属性，元素 <root> 在自己的定义中显式引用自己:

```
<xs:element name="root">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="a" type="xs:string"/>
      <xs:element name="b">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="c" type="xs:string"/>
            <xs:element ref="root" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

对于显式递归，递归分支定界如下:

- 递归分支的开头是元素 Y 的声明，其祖先并未包含另一个 Y 的元素声明。递归分支的开头可有多个子分支；对于特定子分支，如果分支包含另一个 Y 的元素声明，那么该分支被视为递归分支。
- 递归分支的结尾是充当分支开头后代的 Y 的最高级别元素声明。注意，分支的结尾具体的说是元素引用。

作为递归分支开头的节点可充当多个递归分支的起始节点。在以下示例中，有两个显式递归分支:

1. `<root> (*), <b>, <root> (**)`
2. `<root> (*), <b>, <root> (***)`

```
<xs:element name="root"> <!-- * -->
  <xs:complexType>
  <xs:sequence>
    <xs:element name="a" type="xs:string"/>
    <xs:element name="b">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="c" type="xs:string"/>
        <xs:element ref="root" minOccurs="1"/>  <!-- ** -->
        <xs:element ref="root" minOccurs="1"/>  <!-- *** -->
      </xs:sequence>
    </xs:complexType>
    </xs:element>
  </xs:sequence>
  </xs:complexType>
</xs:element>
```

递归分支描述其成员元素如何分解。在实例文档中，对应于递归分支开头的元素 Y 的出现、其后代直到对应于该分支结尾的 Y 的出现都可分解为标量值。实例文档中对应于递归分支的结尾的 Y 的出现标记了递归区域。递归区域开始于此 Y 出现位置的起始元素标记，结束于出现位置的结束元素标记。根据对 db2-xdb:contentHandling 分解注释指定的值，实例文档中包含在此递归区域内的所有元素和属性都可分解为标记或字符串值。

隐式递归

当具有复杂类型定义的元素包含另一元素，同时定义为复杂类型，并且另一元素将所属的复杂类型定义的名称作为其类型属性，则发生隐式递归。如以下示例中所示，元素 <beginRecursion> 引用类型“rootType”，而元素 <beginRecursion> 本身属于要定义的类型“rootType”:

```
<xs:element name="root" type="rootType"/>
<xs:complexType name="rootType">
  <xs:sequence>
  <xs:element name="a" type="xs:string"/>
    <xs:element name="b">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="c" type="xs:string"/>
          <xs:element name="beginRecursion" type="rootType" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

对于隐式递归，递归分支定界如下:

- 递归分支的开头是具有 complexType 类型 CT 的元素 Y 的声明，其祖先并未包含类型为 CT 的另一元素声明。递归分支的开头可有多个子分支；对于特定子分支，如果分支包含另一个类型为 CT 的 Z 的元素声明，那么该分支被视为递归分支。
- 递归分支的结尾是充当分支开头后代的类型为 CT 的最高级别元素声明。

作为递归分支开头的节点可充当多个递归分支的起始节点。在以下示例中，有两个隐式递归分支:

1. `<root>, <b>, <beginRecursion>`
2. `<root>, <b>, <anotherRecursion>`

```
<xs:element name="root" type="rootType"/>
<xs:complexType name="rootType">
  <xs:sequence>
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b">
    <xs:complexType>
    <xs:sequence>
      <xs:element name="c" type="xs:string"/>
      <xs:element name="beginRecursion" type="rootType" minOccurs="2"/>
      <xs:element name="anotherRecursion" type="rootType" minOccurs="0"/>
    </xs:sequence>
    </xs:complexType>
  </xs:element>
  </xs:sequence>
</xs:complexType>
```

与显式递归相比，隐式类型的递归的分解方式有少许差别。在实例文档中，对应于递归分支开头的元素 Y 的出现、其后代直到对应于该分支结尾的 Z 的出现都可分解为标量值。实例文档中 Z 的出现标记递归区域。递归区域从 Z 的起始元素标记之*后*开始，正好到 Z 的结束元素标记之*前*结束。此 Z 的出现位置的所有元素后代都在此递归区域中。但是，此出现位置的属性在递归区域之外，因此会分解为标量值。

递归分支的分解行为

对于两种递归类型，递归分支描述了实例文档对应部分中的非递归和递归区域。只有 XML 实例文档的非递归区域才能分解为目标数据库表中的标量值。此限制包括一个分支内属于封闭分支递归区域的所有非递归区域。即，如果递归分支 RB2 完全包括在递归分支 RB1 中，那么对于实例 XML 文档中 RB2 的某些实例而言，其非递归区域可能落在 RB1 的实例的递归区域内。在此情况下，此非递归区域不能分解为标量值；它是作为 RB1 的分解结果的标记的一部分。对于 RB2 的任何实例，只有不在任何其他递归区域内的实例的非递归区域才能分解为标量值。

例如，以下 XML 模式包含两个递归分支:

1. RB1 (<root> (identified with *), <b>, <root> (identified with **))
2. RB2 (<d>, <d>)

```
<xs:element name="d">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="d">
    </xs:sequence>
    <xs:attribute name="id" type="xs:int"/>
  </xs:complexType>
</xs:element>
<xs:element name="root"> <!-- * -->
  <xs:complexType>
  <xs:sequence>
    <xs:element name="a" type="xs:string"/>
       <xs:element ref="d"/>
    <xs:element name="b">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="c" type="xs:string"/>
        <xs:element ref="root" minOccurs="1"/>  <!-- ** -->
      </xs:sequence>
    </xs:complexType>
    </xs:element>
  </xs:sequence>
  </xs:complexType>
</xs:element>
```

下面突出显示的是关联实例文档的递归区域。实例文档中有两个 RB2 实例（<d> 和 <d>），但只有第一个 RB2 实例（由 # 标识的 <d>）的非递归区域可分解为标量值。即，可分解属性 id="1"。第二个 RB 实例的非递归区域完全在第二个突出显示区域中，它是 RB1 实例的递归区域。因此，不能分解属性 id="2"。

```
<root>
  <a>a str1</a>
  <d id="1"> <d id="11"> </d> </d>
  <b>
    <c>c str1</c>
    <root>
      <a>a str11</a>
      <d id="2"> <d id="22"> </d> </d>
      <b>
      <c>c str11</c>
      </b>
    </root>
  </b>
</root>
```

示例：将 db2-xdb:contentHandling 分解注释与两种类型的递归配合使用

此示例说明显式递归类型和隐式递归类型的分解行为，以及对 db2-xdb:contentHandling 注释设置不同值产生的结果。在以下两个 XML 实例文档中，递归区域已突出显示。

在文档 1 中，当 <root> 元素出现在它自己下面时，递归开始：

```
<root>
  <a>a str1</a>
  <b>
    <c>c str1</c>
    <root>
      <a>a str11</a>
      <b>
        <c>c str11</c>
      </b>
    </root>
  </b>
</root>
```

在文档 2 中，递归从元素 <beginRecursion> 下面的元素开始：

```
<root>
  <a>a str2</a>
  <b>
    <c>c str2</c>
    <beginRecursion>
      <a>a str22</a>
      <b>
        <c>c str22</c>
      </b>
    </beginRecursion>
  </b>
</root>
```

在实例文档中，所有元素或属性及其出现在递归开头和结尾之间的内容不能分解为表－列对中的标量值。但是，可通过对递归分支中的元素（类型为 complexType）添加注释并将 db2-xdb:contentHandling 属性设置为“serializeSubtree”来获取递归开头与结尾之间各项的序列化标记版本。还可通过将 db2-xdb:contentHandling 设置为“stringValue”来获取此部分中所有字符数据的文本序列化。总之，通过对递归分支的任何 complexType 元素或充当递归分支中的元素的祖先的元素设置相应的 db2-xdb:contentHandling 属性，可获取递归路径的内容或标记。

例如，在以下 XML 模式中对元素 <b> 添加注释：

```
<xs:element name="root">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="a" type="xs:string"/>
      <xs:element name="b"
             db2-xdb:rowSet="TABLEx"
             db2-xdb:column="COLx"
             db2-xdb:contentHandling="serializeSubtree">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="c" type="xs:string"/>
            <xs:element ref="root" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
```

```
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

导致分解文档 1 时此 XML 片段插入到行 TABLEx，COLx 中：

```
  <b>
    <c>c str1</c>
    <root>
      <a>a str11</a>
      <b>
        <c>c str11</c>
      </b>
    </root>
  </b>
```

同样，在以下 XML 模式中对元素“beginRecursion”添加注释：

```
<xs:element name="root" type="rootType"/>
<xs:complexType name="rootType">
  <xs:sequence>
    <xs:element name="a" type="xs:string"/>
    <xs:element name="b">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="c" type="xs:string"/>
          <xs:element name="beginRecursion"
            type="rootType" minOccurs="0"
            db2-xdb:rowSet="TABLEx"
            db2-xdb:column="COLx"
            db2-xdb:contentHandling="serializeSubtree"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

导致分解文档 2 时此 XML 片段插入到行 TABLEx，COLx 中：

```
    <beginRecursion>
      <a>a str22</a>
      <b>
        <c>c str22</c>
      </b>
    </beginRecursion>
```

禁用带注释的 XML 模式分解

DB2 在某些情况下可使带注释的 XML 模式分解失效，用户也可以显式地将其禁用。

使分解失效的条件

对于先前已注册并启用了分解的带注释的模式来说，如果满足下列任何条件，就会自动使基于模式的分解失效。（请注意，已使其分解失效的 XML 模式仍可用于分解上下文外部所执行的验证，例如用于 XMLVALIDATE SQL/XML 函数。）对于每个条件，列示了重新启用分解时所需执行的更正操作。

表 51. 使分解失效的条件以及相应的更正操作

条件	用于重新启用分解的操作
注释中引用的表已被废弃	从模式文档中除去对已删除的表的引用，重新注册整个带注释的模式，然后对该模式启用分解
注释中所引用列的数据类型已更改为与 XML 模式类型兼容的类型	通过执行指定了 ENABLE DECOMPOSITION 选项的 ALTER XSROBJECT SQL 语句，对该模式重新启用分解
注释中所引用列的数据类型已更改为与 XML 模式类型不兼容的类型	根据需要调整注释，重新注册整个带注释的模式，然后对该模式启用分解
属于带注释的模式的文档已更改	重新注册所有构成该模式的文档，然后对该模式启用分解

要了解更多信息，请参阅有关注册带注释的模式和启用分解的任务文档。

显式禁用

通过执行下列任何一个 SQL 语句并指定与所要禁用的带注释的模式相对应的 XSR 对象，可以显式地禁用基于模式的分解:

- 指定了 DISABLE DECOMPOSITION 选项的 ALTER XSROBJECT

  注: 禁用了分解的 XML 模式仍可用于验证。
- 指定了 XSROBJECT 选项的 DROP

注: 选择的选项取决于 XML 模式的用途。如果该模式用于验证，那么应该对其禁用分解，而不是将其删除。如果该模式仅用于分解，并且您不想再次将其用于分解，那么可以删除 XSR 对象。

用于带注释的模式分解的 xdbDecompXML 过程

可以通过调用系统定义的 10 个过程的其中一个来调用带注释的 XML 模式分解，这将分解单个 XML 文档。

带注释的 XML 模式分解过程:

- xdbDecompXML
- xdbDecompXML10MB
- xdbDecompXML25MB
- xdbDecompXML50MB
- xdbDecompXML75MB
- xdbDecompXML100MB
- xdbDecompXML500MB
- xdbDecompXML1GB
- xdbDecompXML1_5GB
- xdbDecompXML2GB

这些过程仅仅是 *xmldoc* 参数的大小有所不同，该参数指定要分解的输入文档的大小。请调用大小刚好适合于所要分解的文档大小的过程，以便最大程度地降低系统内存使用量。例如，要分解大小为 1MB 的文档，请使用 xdbDecompXML 过程。

下面提供了 xdbDecompXML 的语法；请参阅 *xmldoc* 参数的描述以了解针对 xdbDecompXML10MB、xdbDecompXML25MB、xdbDecompXML50MB、xdbDecompXML75MB、xdbDecompXML100MB、xdbDecompXML500MB、xdbDecompXML1GB、xdbDecompXML1_5GB 和 xdbDecompXML2GB 过程的 *xmldoc* 参数规范。

语法

```
►►—xdbDecompXML—(—rschema—,—xmlschemaname—,—xmldoc—,—documentid—,——————►

►—validation—,—reserved—,—reserved—,—reserved—)——————————————————————————►◄
```

此过程的模式为 SYSPROC。

权限

属于调用此过程的语句的授权标识必须拥有下列其中一项特权或权限:

- 需要下列所有特权:
  - 对带注释的模式中引用的所有目标表的 INSERT 特权
  - 对 db2-xdb:expression 或 db2-xdb:condition 注释引用的任何表的 SELECT、INSERT、UPDATE 或 DELETE 特权（如果适用）
- 对带注释的模式文档集中引用的所有目标表的 CONTROL 特权
- DATAACCESS 权限

如果 *validation* 的值为 1，那么属于调用此过程的语句的授权标识必须对 XML 模式具备 USAGE 特权。

过程参数

rschema

VARCHAR(128) 类型的输入参数，它指定向 XML 模式存储库注册的两部分 XSR 对象名的 SQL 模式部分。如果此值为 `NULL`，那么假定 SQL 模式部分是 CURRENT SCHEMA 专用寄存器的当前值。

xmlschemaname

VARCHAR(128) 类型的输入参数，它指定向 XML 模式存储库注册的两部分 XSR 对象名的模式名。此值不能为 `NULL`。

xmldoc

BLOB(1M) 类型的输入参数，它指定包含所要分解的 XML 文档的缓冲区。

注:

- 对于 xdbDecompXML10MB 过程，此参数的类型为 BLOB(10M)。
- 对于 xdbDecompXML25MB 过程，此参数的类型为 BLOB(25M)。
- 对于 xdbDecompXML50MB 过程，此参数的类型为 BLOB(50M)。
- 对于 xdbDecompXML75MB 过程，此参数的类型为 BLOB(75M)。

- 对于 xdbDecompXML100MB 过程，此参数的类型为 BLOB(100M)。
- 对于 xdbDecompXML500MB 过程，此参数的类型为 BLOB(500M)。
- 对于 xdbDecompXML1GB 过程，此参数的类型为 BLOB(1G)。
- 对于 xdbDecompXML1_5GB 过程，此参数的类型为 BLOB(1.5G)。
- 对于 xdbDecompXML2GB 过程，此参数的类型为 BLOB(2G)。

documentid

VARCHAR(1024) 类型的输入参数，它指定所要分解的输入 XML 文档的标识。此参数中提供的值将替换相应 XML 模式中的 db2-xdb:expression 或 db2-xdb:condition 注释中指定的 $DECOMP_DOCUMENTID。

validation

INTEGER 类型的输入参数，它指示是否对所分解的文档执行验证。可能的值包括:

0 在分解输入文档前不对其执行验证。

1 根据先前向 XML 模式存储库注册的 DTD 或 XML 模式文档来对输入文档执行验证。只有验证成功后才分解输入 XML 文档。

reserved

reserved 参数是为将来使用而保留的输入参数。为这些参数传递的值必须是 NULL。

输出

此过程没有显式的输出参数。请检查 SQLCA 结构的 sqlcode 字段以了解分解期间发生的任何错误。完成分解后，可能的 sqlcode 值如下所示:

0 已成功地分解了文档。

正整数

已成功地分解了文档，但发生了警告情况。这些警告记录在 db2diag 日志文件中，该文件位于首次出现数据捕获（FODC）存储目录中。

负整数

未能分解文档。sqlcode 指示了故障原因。请检查 db2diag 日志文件以了解有关故障的详细信息。

使用说明

这些过程在使用读稳定性隔离级别的情况下执行。

这些过程以原子方式运行；如果过程在执行期间失败，就会回滚此过程执行的所有操作。由于此过程本身并不执行 COMMIT SQL 语句，因此，要落实此过程所作的更改，调用者必须执行 COMMIT 语句。

如果正在使用脚本或应用程序来分解若干个未知大小的文档，请考虑使用 DECOMPOSE XML DOCUMENT 命令来进行分解，而不要使用 xdbDecompXML 过程，原因是该命令会自动调用适合于文档大小的过程。

示例

已向 XML 模式存储库将带注释的 XML 模式注册为 ABC.TEST，该模式指示 XML 值所要分解为的表和列。要将 XML 文档分解到关系表中，请按如下调用适当的过程:

```
CALL xdbDecompXML ('ABC', 'TEST', BLOB('<Element1>Hello world</Element1>'),
     'DOCID', 0 , NULL, NULL, NULL)
```

DECOMPOSE XML DOCUMENT

此命令将调用存储过程来分解使用已注册并且支持分解的 XML 模式的单个 XML 文档。

权限

需要下面的其中一组特权或权限:

- 下列其中一项权限:
  - 对带注释的模式文档集中引用的所有目标表的 CONTROL 特权
  - DATAACCESS 权限
- 下列所有特权:
  - 对目标表的 INSERT 特权，就像操作文件中指定的操作所需要的特权一样
  - 对 db2-xdb:expression 或 db2-xdb:condition 注释引用的任何表的 SELECT、INSERT、UPDATE 或 DELETE 特权（如果适用）

如果指定了 VALIDATE 选项，那么还需要对 XML 模式的 USAGE 特权。

必需的连接

Database

命令语法

```
>>—DECOMPOSE XML DOCUMENT—xml-document-name—XMLSCHEMA—xml-schema-name——————>

>—+——————————+——————————————————————————————————————————————————————————————><
  └─VALIDATE─┘
```

命令参数

DECOMPOSE XML DOCUMENT *xml-document-name*

xml-document-name 是要分解的输入 XML 文档的文件路径和文件名。

XMLSCHEMA *xml-schema-name*

xml-schema-name 是已向 XML 模式存储库注册的、要用于文档分解的现有 XML 模式的名称。*xml-schema-name* 是一个限定 SQL 标识，它包括一个可选的 SQL 模式名、该模式名后跟的一个句点和 XML 模式名。如果未指定 SQL 模式名，那么假定它是 DB2 专用寄存器 CURRENT SCHEMA 的值。

VALIDATE

此参数指示首先要验证输入 XML 文档，仅当该文档有效的情况下才进行分解。如果未指定 VALIDATE，那么在进行分解之前不会验证输入 XML 文档。

示例

以下示例指定要使用已注册的 XML 模式 DB2INST1.GENBANKSCHEMA 对其进行验证和分解的 XML 文档 ./gb/document1.xml。

```
DECOMPOSE XML DOCUMENT ./gb/document1.xml
                          XMLSCHEMA DB2INST1.GENBANKSCHEMA
             VALIDATE
```

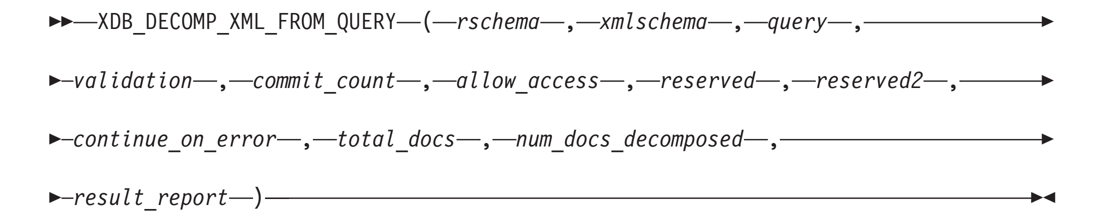
以下示例指定要分解 XML 文档 ./gb/document2.xml，但不会使用已注册的 XML 模式 DB2INST2."GENBANK SCHEMA1" 进行验证，并且假定 DB2 专用寄存器 CURRENT SCHEMA 的值设置为 DB2INST2。

```
DECOMPOSE XML DOCUMENT ./gb/document2.xml
                              XMLSCHEMA "GENBANK SCHEMA1"
```

带注释的模式分解的 XDB_DECOMP_XML_FROM_QUERY 存储过程

此存储过程用于分解二进制文件或 XML 列中的一个或多个 XML 文档。XML 文档数据根据 XML 模式中所指定的注释来存储在关系表的各列中。

语法



此存储过程的模式为 SYSPROC。

此过程在使用读稳定性隔离级别的情况下执行。

权限

需要以下其中一项权限或特权:

- 需要下列所有特权:
  - 对带注释的模式中引用的所有目标表的 INSERT 特权
  - 对于具有输入文档的列，您需要对包含此列的表、别名或视图具备 SELECT 特权
  - 对 db2-xdb:expression 或 db2-xdb:condition 注释引用的任何表的 SELECT、INSERT、UPDATE 或 DELETE 特权（如果适用）
- 对于带注释的模式文档集中引用的所有表具有 CONTROL 特权；并且对于具有输入文档的列，您需要对包含此列的表、别名或视图也具备 CONTROL 特权。
- DATAACCESS 权限

如果 *validation* 的值为 1，那么还需要对 XML 模式具备 USAGE 特权。

过程参数

rschema
: VARCHAR(128) 类型的输入参数，它指定向 XML 模式存储库注册的两部分 XSR 对象名的 SQL 模式部分。此值可为 NULL。如果此值为 NULL，那么假定 SQL 模式部分是 CURRENT SCHEMA 专用寄存器的当前值。

xmlschema
: VARCHAR(128) 类型的输入参数，它指定向 XSR 注册的两部分 XSR 对象名的名称。此值不能为 NULL。

query
: CLOB(1MB) 类型的输入参数。此值不能为 NULL。*query* 符合 SQL SELECT 语

句的规则，并且必须返回一个包含两列的结果集。第一列是文档标识。每个文档标识将唯一地标识要分解的 XML 文档。此列必须是字符类型或者可转换为字符类型。第二列包含要分解的 XML 文档。文档列的类型可为 XML、BLOB 和 VARCHAR FOR BIT DATA。

例如，以下 SELECT 语句中的 DOCID 列包含存储在 SALESDOC 列中的 XML 文档的唯一标识。

```
SELECT DOCID, SALESDOC FROM SALESTAB
```

validation

INTEGER 类型的输入参数，它指示是否在分解文档之前对文档执行验证。可能的值包括:

0 在分解输入文档之前不对这些文档执行验证。

如果传递的值为 0 并且未执行验证，那么在调用存储过程之前将由用户负责验证文档。例如，将 XML 文档插入列中时，用户可以使用 XMLVALIDATE，或者在插入文档之前使用 XML 处理器。如果输入 XML 文档无效并且为此参数指定的值为 0，那么分解结果未定义。

1 将针对先前已向 XML 模式存储库注册的 DTD 或 XML 模式文档来对输入文档执行验证。仅当验证成功之后才会分解输入 XML 文档。

commit_count

类型为 INTEGER 的输入参数。可能的值包括:

0 存储过程未执行过 COMMIT 语句。

n（**一个正整数**）

每当 *n* 次对文档成功进行分解之后，就会执行 COMMIT 语句。

allow_access

类型为 INTEGER 的输入参数。可能的值包括:

0 对于具有 XML 模式中的映射的所有表，存储过程将获得对于这些表的互斥锁定（X）。在分解每个文档期间，并不是所有表都必需参与分解，但是所有目标表都将被锁定，以降低在长的工作单元中发生死锁的可能性。

1 当获取锁定时，存储过程将等待，并且有可能超时。

reserved

reserved 参数是保留给将来使用的输入参数。为此参数传递的值必须为 NULL。

reserved2

reserved2 参数是保留给将来使用的输入参数。为此参数传递的值必须为 NULL。

continue_on_error

类型为 INTEGER 的输入参数。可能的值包括:

0 在无法成功分解的第一个文档中，存储过程将停止。如果在分解文档期间发生了错误，那么将撤销分解此文档期间对数据库所作的更改。

1 发生特定于文档的错误时，存储过程不会停止，并且会尝试分解由 *query* 指定的所有文档。如果在分解文档期间发生了错误，那么将撤销分解此文档期间对数据库所作的更改，并且存储过程将尝试分解下一个文档。有关未成功分解的任何文档的信息都将写入 *result_report*。

无论 *continue_on_error* 的值如何，每当发生致命错误以及非特定于文档的错误时，存储过程都不会继续执行。

total_docs
类型为 INTEGER 的输出参数，用于指定 XDB_DECOMP_XML_FROM_QUERY 存储过程已尝试分解的输入文档总数。

num_docs_decomposed
类型为 INTEGER 的输出参数，用于指定已成功分解的文档数。

result_report
类型为 BLOB(100MB) 的输出参数。缓冲区中包含一个 UTF-8 XML 文档，此文档列示了未成功分解的每个输入文件的名称和一条诊断消息。仅当至少有一个未能成功分解的 XML 文档时，才会生成此报告。

result_report 中的 XML 文档的格式为如下所示:

```
<?xml version='1.0'?>
<xdb:errorReport xmlns:xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
   <xdb:document>
      <xdb:documentId>sssss</xdb:documentId>
      <xdb:errorMsg>qqqqq</xdb:errorMsg>
   </xdb:document>
   <xdb:document>
      . . .
   </xdb:document>
   . . .
</xdb:errorReport>
```

文档标识值 *sssss* 是由 *query* 指定的第一列中的值。此值标识未成功分解的 XML 文档。错误消息值 *qqqqq* 是在尝试分解文档期间遇到的错误。

输出

SQLCA 结构指示在尝试分解 XML 文档之后过程的返回状态。过程可能会返回下列其中一个 SQLCODE 值:

0 成功分解了由 *query* 指定的所有文档。

16278
分解一个或多个文档失败。成功分解的文档数由存储过程的 *num_docs_decomposed* 输出参数给定。每个失败的文档单独的错误消息记录在 *result_report* 中。db2diag 日志文件中记录了有关每个故障的更多诊断详细信息。

负整数
未分解任何文档。SQLCODE 指示故障原因。请检查 db2diag 日志文件以了解有关故障的详细信息。

注意

存储过程借助下列特征来进行声明:

```
DYNAMIC RESULT SETS 0
NOT DETERMINISTIC
UNFENCED
THREADSAFE
MODIFIES SQL DATA
PARAMETERSTYLE SQL
CALLED ON NULL INPUT
NEW SAVEPOINT LEVEL
DBINFO
```

示例

以下示例假定 ABC.SALESTAB 表中包含 SALESDOC 和 DOCID 这两列。SALESDOC 列包含 XML 文档，而 DOCID 列包含 SALESDOC 中的 XML 文档的唯一标识。所有 XML 文档对应于作为 ABC.SALES 注册的 XML 模式，已使用分解信息注释了此模式并且支持分解。以下示例将调用此存储过程，以使用 ABC.SALES 模式来分解存储在 SALESDOC 中的文档:

```
XDB_DECOMP_XML_FROM_QUERY ('ABC', 'SALES',
    'SELECT DOCID, SALESDOC FROM ABC.SALESTAB', 0, 0, 0,
    NULL, NULL, 1, numInput, numDecomposed, errorreportBuf);
```

XML 分解注释

带注释的 XML 模式分解根据 XML 模式文档中添加的注释进行。这些分解注释充当 XML 文档元素或属性与数据库中目标表和列之间的映射。分解处理引用这些注释以确定 XML 文档分解方式。

XML 分解注释属于 http://www.ibm.com/xmlns/prod/db2/xdb1 名称空间，在文档中，这些注释由“db2-xdb”前缀标识。您可以选择自己的前缀；但是，如果这样做，就必须将该前缀与以下名称空间绑定：http://www.ibm.com/xmlns/prod/db2/xdb1。分解过程仅识别对该 XML 模式启用分解时在此名称空间下的注释。

仅当将分解注释添加到模式文档中的元素和属性声明或者作为全局注释添加它们时，分解过程才能识别它们。它们是作为元素或属性声明的 <xs:annotation> 子元素的属性或组成部分指定的。添加到复杂类型、引用或其他 XML 模式构造中的注释将被忽略。

虽然包含在 XML 模式文档中，但这些注释并不会影响模式文档的原始结构，也不会参与 XML 文档的验证。它们仅由 XML 分解过程引用。

作为分解过程的核心功能的两个注释为：db2-xdb:rowSet 和 db2-xdb:column。这两个注释分别指定已分解的值的目标表和列。必须指定这两个注释，分解过程才能成功完成。其他注释是可选的，但可用于进一步控制分解过程的操作。

XML 分解注释 － 规范和作用域

可以在 XML 模式文档中将用于分解的注释指定为元素或属性声明。

可以将注释指定为:

- 元素或属性声明中的简单属性，或
- 元素或属性声明的结构化（复杂）子元素，它由简单元素和属性组成

作为属性的注释

在元素或属性声明中指定为简单属性的注释只适用于指定它的元素或属性。

例如，可以将 db2-xdb:rowSet 和 db2-xdb:column 分解注释指定为属性。按如下所示指定这些注释:

```
<xs:element name="isbn" type="xs:string"
            db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="ISBN"/>
```

db2-xdb:rowSet 和 db2-xdb:column 注释仅适用于名为 ISBN 的这个元素。

作为结构化子元素的注释

必须在模式文档中 XML 模式所定义 <xs:annotation><xs:appinfo></xs:appinfo></xs:annotation> 层次结构内指定要作为元素或属性声明的结构化子元素的注释。

例如，可按如下所示将 db2-xdb:rowSet 和 db2-xdb:column 注释指定为子元素（它们是 <db2-xdb:rowSetMapping> 注释的子元素）：

```
<xs:element name="isbn" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>TEXTBOOKS</db2-xdb:rowSet>
        <db2-xdb:column>ISBN</db2-xdb:column>
      </db2-xdb:rowSetMapping>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

将 db2-xdb:rowSet 和 db2-xdb:column 注释指定为子元素与将这些注释指定为属性相同（在前面已经说明了这一点）。与将注释指定为属性的方法相比，虽然将注释指定为子元素要复杂很多，但在需要对一个元素或属性指定多个 <db2-xdb:rowSetMapping> 时（即，需要在同一元素或属性声明中指定多个映射时），还是需要将注释指定为子元素。

全局注释

当注释被指定为 <xs:schema> 元素的子元素时，它就是适用于组成 XML 模式的所有 XML 模式文档的全局注释。

例如，<db2-xdb:defaultSQLSchema> 注释指示 XML 模式中引用的所有未限定表的缺省 SQL 模式。必须将 <db2-xdb:defaultSQLSchema> 指定为 <xs:schema> 的子元素:

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>admin</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

此声明指定组成此 XML 模式的所有模式文档内的所有未限定的表将具有 SQL 模式 “admin”。

请参阅具体注释的文档以确定可以如何指定特定注释。

XML 分解注释 － 总结

DB2 支持由带注释的 XML 模式分解过程使用的一组注释，该组注释用于将 XML 文档中的元素和属性映射至目标数据库表。下面对一些 XML 分解注释作了概括，这些总结按照您使用注释来执行的任务和操作进行分组。

要了解有关特定注释的更多信息，请参阅有关该注释的详细文档。

表 52. 指定 SQL 模式

操作	XML 分解注释
为所有未指定 SQL 模式的表指定缺省 SQL 模式	db2-xdb:defaultSQLSchema
为特定的表指定不同于缺省 SQL 模式的 SQL 模式	db2-xdb:table（<db2-xdb:SQLSchema> 子元素）

表 53. 将 XML 元素或属性映射至目标基本表

操作	XML 分解注释
将单个元素或属性映射至单个表/列对	db2-xdb:rowSet（指定 db2-xdb:column 作为属性注释）或 db2-xdb:rowSetMapping
将单个元素或属性映射至一个或多个独特的表/列对	db2-xdb:rowSetMapping
将多个元素或属性映射至单个表/列对	db2-xdb:table
指定目标表之间的排序依赖性	db2-xdb:rowSetOperationOrder，db2-xdb:rowSet，db2-xdb:order

表 54. 指定要分解的 XML 数据

操作	XML 分解注释
指定要为复杂类型元素插入的内容类型（文本、字符串或标记）	db2-xdb:contentHandling
指定插入内容前要执行的任何内容变换操作	db2-xdb:normalization，db2-xdb:expression，db2-xdb:truncate
根据项的内容或它所在的上下文对要分解的数据进行过滤	db2-xdb:condition db2-xdb:locationPath

db2-xdb:defaultSQLSchema 分解注释

db2-xdb:defaultSQLSchema 注释指定在未使用 db2-xdb:table 注释进行显式限定的情况下，XML 模式中引用的所有表名的缺省 SQL 模式。

<db2-xdb:defaultSQLSchema> 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

作为全局 <xs:annotation> 元素子代的 <xs:appinfo> 的子元素。

如何指定

通过以下方法来指定 db2-xdb:defaultSQLSchema（其中 *value* 表示有效注释值）：

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>value</db2-xdb:defaultSQLSchema>
```

```
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效值

普通 SQL 模式名或定界 SQL 模式名。普通 SQL 模式名（即，未定界 SQL 模式名）不区分大小写。要指定定界 SQL 模式，请使用通常用来对 SQL 标识进行定界的引号。在 XML 模式文档中，必须对包含特殊字符“<”和“&”的 SQL 模式名进行转义。

详细信息

对于带注释的模式中引用的所有表来说，必须使用它们的 SQL 模式来对它们进行限定。可以通过两种方法对表进行限定：显式地指定 <db2-xdb:table> 注释的 <db2-xdb:SQLSchema> 子元素，或者使用 <db2-xdb:defaultSQLSchema> 全局注释。对于任何未限定表名来说，将 <db2-xdb:defaultSQLSchema> 中指定的值用作它的 SQL 模式名。如果在一个带注释的模式中有多个模式文档指定了此注释，那么所有值必须相同。

示例

以下示例说明如何将普通 SQL 标识（即，未定界 SQL 标识）admin 定义成带注释的模式中所有未限定表的 SQL 模式：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>admin</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

以下示例说明如何将定界 SQL 标识 admin schema 定义成带注释的模式中所有未限定表的 SQL 模式。请注意，必须使用引号来对 admin schema 进行定界。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>"admin schema"</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

db2-xdb:rowSet 分解注释

db2-xdb:rowSet 注释指定 XML 元素或属性到目标基本表的映射。

db2-xdb:rowSet 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

<xs:element> 或 <xs:attribute> 的属性，或者 <db2-xdb:rowSetMapping> 或 <db2-xdb:order> 的子元素。

如何指定

通过下列任何一种方法来指定 db2-xdb:rowSet（其中 *value* 表示有效注释值）：

- `<xs:element db2-xdb:rowSet="value" />`
- `<xs:attribute db2-xdb:rowSet="value" />`
- ```
  <db2-xdb:rowSetMapping>
    <db2-xdb:rowSet>value</db2-xdb:rowSet>
    ...
  </db2-xdb:rowSetMapping>
  ```
- ```
  <db2-xdb:order>
    <db2-xdb:rowSet>value</db2-xdb:rowSet>
    ...
  </db2-xdb:order>
  ```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效值

任何符合 SQL 标识规则的标识。有关更多信息，请参阅标识文档。

详细信息

db2-xdb:rowSet 注释将 XML 元素或属性映射至目标基本表。此注释可以直接标识表名，也可以标识较为复杂的映射中的行集名（在这种情况下，通过 db2-xdb:table 注释使该行集与表名相关联）。在简单映射中，此注释指定要将值分解到的表的名称。在较为复杂的映射中，多个行集（每个行集的名称互不相同）映射至同一个表，所以此注释指定行集而不是表名。

此 XML 元素值或属性值将被分解到的目标基本表是由构成带注释的模式的模式文档集中其他注释确定的:

- 如果 db2-xdb:rowSet 值与 <db2-xdb:table> 全局注释的任何 <db2-xdb:rowSet> 子元素都不匹配，那么目标表名是此注释指定的值，并且使用 <db2-xdb:defaultSQLSchema> 全局注释定义的 SQL 模式进行限定。db2-xdb:rowSet 的这种用法用于以下情况：对于特定的表，只有一组元素或属性映射至该表。
- 如果 db2-xdb:rowSet 值与 <db2-xdb:table> 全局注释的 <db2-xdb:rowSet> 子元素匹配，那么目标表名是 <db2-xdb:table> 的 <db2-xdb:name> 子代中指定的表名。db2-xdb:rowSet 的这种用法用于较为复杂的情况，即，对于特定的表，有多组可能相互重叠的元素或属性映射至该表。

重要事项: 在向 XML 模式存储库注册 XML 模式时，确保此注释引用的表存在于数据库中。（在注册 XML 模式时，db2-xdb:column 注释中指定的列也必须存在。）如果该表不存在，对该 XML 模式启用分解时就会返回错误。如果 <db2-xdb:table> 指定了除表以外的对象，也会返回错误。

使用 db2-xdb:rowSet 注释时，必须指定 db2-xdb:column 注释或 db2-xdb:condition 注释。db2-xdb:rowSet 与 db2-xdb:column 共同描述此元素或属性将分解到的表和列。db2-xdb:rowSet 与 db2-xdb:condition 的组合指定一个条件，该条件必须成立，这样才能将该行集的任何行插入到表中（直接引用该表，或者通过 <db2-xdb:table> 注释间接地引用该表）。

示例

上面列示了两种使用 db2-xdb:rowSet 的方法，下面对这两种方法进行说明。

单组元素或属性映射至同一个表

对于以下带注释的模式部分，假定 BOOKCONTENTS 表属于 <db2-xdb:defaultSQLSchema> 指定的 SQL 模式，并假定不存在 <db2-xdb:rowSet> 子元素与“BOOKCONTENTS”匹配的全局 <db2-xdb:table> 元素。

```
 <xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer" />
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string"
                db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="ISBN" />
    <xs:attribute name="title" type="xs:string" />
  </xs:complexType>
</xs:element>

<xs:complexType name="chapterType">
  <xs:sequence>
    <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTCONTENT" />
  </xs:sequence>
  <xs:attribute name="number" type="xs:integer"
                db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTNUM" />
  <xs:attribute name="title" type="xs:string"
                db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTTITLE" />
</xs:complexType>

<xs:simpleType name="paragraphType">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

考虑 XML 文档中的以下元素:

```
<book isbn="1-11-111111-1" title="My First XML Book">
  <authorID>22</authorID>
  <!-- this book does not have a preface -->
  <chapter number="1" title="Introduction to XML">
    <paragraph>XML is fun...</paragraph>
    ...
  </chapter>
  <chapter number="2" title="XML and Databases">
    <paragraph>XML can be used with...</paragraph>
  </chapter>
  ...
  <chapter number="10" title="Further Reading">
    <paragraph>Recommended tutorials...</paragraph>
  </chapter>
  ...
</book>
```

于是，BOOKCONTENTS 表中填充的内容如下所示:

表 55. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	Introduction to XML	XML is fun...
1-11-111111-1	2	XML and Databases	XML can be used with...
...	...	...	...
1-11-111111-1	10	Further Reading	Recommended tutorials...

多组元素或属性映射至同一个表

如果 <db2-xdb:table> 全局注释包含与 db2-xdb:rowSet 注释中指定的值相匹配的 <db2-xdb:rowSet> 子元素，那么通过 <db2-xdb:table> 注释将该元素或属性映射至表。对于以下带注释的模式部分，假定 ALLBOOKS 表属于 <db2-xdb:defaultSQLSchema> 指定的 SQL 模式。

```
<!-- global annotation -->
<xs:annotation>
  <xs:appinfo>
    <db2-xdb:table>
       <db2-xdb:name>ALLBOOKS</db2-xdb:name>
       <db2-xdb:rowSet>book</db2-xdb:rowSet>
       <db2-xdb:rowSet>textbook</db2-xdb:rowSet>
    </db2-xdb:table>
  </xs:appinfo>
</xs:annotation>

  <xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer"
                  db2-xdb:rowSet="book" db2-xdb:column="AUTHORID" />
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string"
                  db2-xdb:rowSet="book" db2-xdb:column="ISBN" />
    <xs:attribute name="title" type="xs:string"
                  db2-xdb:rowSet="book" db2-xdb:column="TITLE" />
  </xs:complexType>
</xs:element>
<xs:element name="textbook">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isbn" type="xs:string"
                  db2-xdb:rowSet="textbook" db2-xdb:column="ISBN" />
      <xs:element name="title" type="xs:string"
                  db2-xdb:rowSet="textbook" db2-xdb:column="TITLE" />
      <xs:element name="primaryauthorID" type="xs:integer"
                  db2-xdb:rowSet="textbook" db2-xdb:column="AUTHORID" />
      <xs:element name="coauthorID" type="xs:integer"
                  minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="subject" type="xs:string" />
      <xs:element name="edition" type="xs:integer" />
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="chapterType">
```

```
  <xs:sequence>
    <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="number" type="xs:integer" />
  <xs:attribute name="title" type="xs:string" />
</xs:complexType>

<xs:simpleType name="paragraphType">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

考虑 XML 文档中的下列元素:

```
<book isbn="1-11-111111-1" title="My First XML Book">
  <authorID>22</authorID>
  <!-- this book does not have a preface -->
  <chapter number="1" title="Introduction to XML">
    <paragraph>XML is fun...</paragraph>
  </chapter>
  <chapter number="2" title="XML and Databases">
    <paragraph>XML can be used with...</paragraph>
  </chapter>
  <chapter number="10" title="Further Reading">
    <paragraph>Recommended tutorials...</paragraph>
  </chapter>
</book>

<textbook>
  <isbn>0-11-011111-0</isbn>
  <title>Programming with XML</title>
  <primaryauthorID>435</primaryauthorID>
  <subject>Programming</subject>
  <edition>4</edition>
  <chapter number="1" title="Programming Basics">
    <paragraph>Before you being programming...</paragraph>
  </chapter>
  <chapter number="2" title="Writing a Program">
    <paragraph>Now that you have learned the basics...</paragraph>
  </chapter>
  ...
  <chapter number="10" title="Advanced techniques">
    <paragraph>You can apply advanced techniques...</paragraph>
  </chapter>
</textbook>
```

在本示例中，有两组元素或属性映射至表 ALLBOOKS:

- /book/@isbn、/book/@authorID 和 /book/title
- /textbook/isbn、/textbook/primaryauthorID 和 /textbook/title

这两组元素或属性是通过与不同的行集名称相关联来区分的。

表 56. ALLBOOKS

ISBN	TITLE	AUTHORID
1-11-111111-1	My First XML Book	22
0-11-011111-0	Programming with XML	435

db2-xdb:table 分解注释

<db2-xdb:table> 注释将多个 XML 元素或属性映射至同一个目标列；您也可以使用该注释来指定具有 SQL 模式的目标表，该 SQL 模式不同于 <db2-xdb:defaultSQLSchema> 所指定的缺省 SQL 模式。

<db2-xdb:table> 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

<xs:appinfo>（这是 <xs:annotation> 的子元素）的全局子元素

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效结构

受支持的 <db2-xdb:table> 子元素如下所示，按照指定它们时必须遵循的顺序进行列示:

<db2-xdb:SQLSchema>
: （可选）表的 SQL 模式。

<db2-xdb:name>
: 基本表的名称。当使用上述 <db2-xdb:SQLSchema> 注释值或 <db2-xdb:defaultSQLSchema> 注释值对此表名进行限定后，它在构成带注释的模式的 XML 模式文档集的所有 <db2-xdb:table> 注释中必须是唯一的。

<db2-xdb:rowSet>
: 所有指定了相同 <db2-xdb:rowSet> 值的元素和属性构成一行。由于可以对同一个 <db2-xdb:name> 值指定多个 <db2-xdb:rowSet> 元素，所以可以有多组映射与单个表相关联。<db2-xdb:rowSet> 值与 db2-xdb:column 注释中指定的列的组合允许将单个 XML 文档中的多组元素或属性映射至同一个表的列。

  必须至少指定一个 <db2-xdb:rowSet> 元素，并且每个 <db2-xdb:rowSet> 元素在构成带注释的模式的 XML 模式文档集的所有 <db2-xdb:table> 注释中都必须是唯一的，这样该注释才有效。

在 <db2-xdb:table> 的子元素的字符内容中，空格是有意义的，不会对其进行规范化。这些元素的内容必须遵循 SQL 标识的拼写规则。未定界值不区分大小写；对于定界值来说，将引号用作定界符。必须对包含特殊字符“<”和“&”的 SQL 标识进行转义。

详细信息

在下列任何一种情况下，必须使用 <db2-xdb:table> 注释:

- 在将多个祖代行映射至同一个表列时（涉及单一位置路径的映射（这表示该表只有一组列映射）无需使用此注释；但是，需要使用 db2-xdb:rowSet 注释）
- 当用于存放所分解数据的表的模式与 <db2-xdb:defaultSQLSchema> 注释定义的 SQL 模式不同时。

只能指定基本表；此映射不支持其他类型的表（例如类型表、总结表、临时表或具体化查询表）。只能对 DB2 数据库 Linux 版、UNIX 版和 Windows 版数据源对象指定昵称。目前不允许此注释使用视图和表别名。

示例

以下示例说明在将多个位置路径映射至同一列时，如何使用 <db2-xdb:table> 注释来将相关元素和属性分组到一起以构成一行。首先，考虑 XML 文档中的下列元素（对用于其他注释的示例作了小幅修改）。

```
<root>
  ...
  <book isbn="1-11-111111-1" title="My First XML Book">
      <authorID>22</authorID>
        <email>author22@anyemail.com</email>
      <!-- this book does not have a preface -->
      <chapter number="1" title="Introduction to XML">
          <paragraph>XML is fun...</paragraph>
      ...
      </chapter>
      <chapter number="2" title="XML and Databases">
          <paragraph>XML can be used with...</paragraph>
      </chapter>
    ...
      <chapter number="10" title="Further Reading">
          <paragraph>Recommended tutorials...</paragraph>
      </chapter>
    </book>  ...
    <author ID="0800" email="author800@email.com">
      <firstname>Alexander</firstname>
      <lastname>Smith</lastname>
      <activeStatus>0</activeStatus>
    </author>
  ...
<root>
```

假定此分解映射的用途是将包含作者标识及其相应电子邮件地址的行插入到同一个 AUTHORSCONTACT 表中。请注意，<book> 元素和 <author> 元素都包含作者标识和电子邮件地址。因此，需要将多个位置路径映射至同一个表中相同的列。所以，必须使用 <db2-xdb:table> 注释。下面是带注释的模式中的部分内容，此部分内容说明如何使用 <db2-xdb:table> 来使多个路径与同一个表相关联。

```
<!-- global annotation -->
          <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>adminSchema</db2-xdb:defaultSQLSchema>
      <db2-xdb:table>
        <db2-xdb:SQLSchema>user1</db2-xdb:SQLSchema>
        <db2-xdb:name>AUTHORSCONTACT</db2-xdb:name>
        <db2-xdb:rowSet>bookRowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>authorRowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="authorID" type="xs:integer"
                    db2-xdb:rowSet="bookRowSet" db2-xdb:column="AUTHID" />
        <xs:element name="email" type="xs:string"
                    db2-xdb:rowSet="bookRowSet" db2-xdb:column="EMAILADDR" />
        <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string" />
      <xs:attribute name="title" type="xs:string" />
    </xs:complexType>
  </xs:element>
```

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string" />
      <xs:element name="lastname" type="xs:string" />
      <xs:element name="activeStatus" type="xs:boolean" />
    </xs:sequence>
    <xs:attribute name="ID" type="xs:integer"
                  db2-xdb:rowSet="authorRowSet" db2-xdb:column="AUTHID" />
    <xs:attribute name="email" type="xs:string"
                  db2-xdb:rowSet="authorRowSet" db2-xdb:column="EMAILADDR" />
  </xs:complexType>
</xs:element>
```

<db2-xdb:table> 注释标识要与 db2-xdb:name 子元素映射的目标表的名称。在本示例中，AUTHORSCONTACT 是目标表。为了确保 <book> 元素中的标识和电子邮件地址与 <author> 元素中的那些内容分开存放（即，每一行包含逻辑上相关的值），使用了 <db2-xdb:rowSet> 元素来使相关项相互关联。虽然本示例中的 <book> 和 <author> 元素是独立的实体，但在某些情况下，要映射的实体不是独立的，需要对它们进行逻辑分隔，这可以通过使用行集实现。

请注意，AUTHORSCONTACT 表所在的 SQL 模式不是缺省 SQL 模式，使用了 <db2-xdb:SQLSchema> 元素来指定这种情况。得到的 AUTHORSCONTACT 表如下所示：

表 57. AUTHORSCONTACT

AUTHID	EMAILADDR
22	author22@anyemail.com
0800	author800@email.com

本示例说明如何通过行集对值进行逻辑分组，从而确保不会无意中将不相关的值映射至同一表/列对。在本示例中，/root/book/authorID 与 /root/author/@ID 映射至同一表/列对。同样，/root/book/email 与 /root/author/@email 映射至同一表/列对。请考虑没有行集可用的情况。例如，如果在 <author> 元素实例中不存在 /root/book/email 元素，并且无法使用行集，那么不能确定 <author> 元素中的电子邮件是应该与 /root/book/authorID 还是 /root/author/@ID 相关联（或者同时与这两者相关联）。因此，通过在 <db2-xdb:table> 注释中使行集与单个表相关联，有助于在逻辑上对不同行集进行区分。

db2-xdb:column 分解注释

db2-xdb:column 注释指定 XML 元素或属性所映射至的表列名。

db2-xdb:column 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

<xs:element> 或 <xs:attribute> 的属性，或者 <db2-xdb:rowSetMapping> 的子元素。

如何指定

通过下列任何一种方法来指定 db2-xdb:column（其中 *value* 表示有效注释值）：

- `<xs:element db2-xdb:rowSet="value" db2-xdb:column="value" />`

- `<xs:attribute db2-xdb:rowSet="value" db2-xdb:column="value" />`
-
```
<db2-xdb:rowSetMapping>
  <db2-xdb:rowSet>value</db2-xdb:rowSet>
  <db2-xdb:column>value</db2-xdb:column>
  ...
</db2-xdb:rowSetMapping>
```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效值

任何符合下列条件的基本表列名:

- 未定界的列名不区分大小写。对于定界的列名来说，使用 `"` 对定界符进行转义。例如，要指定由两个单词组成的列名“col one”，db2-xdb:column 的设置如下所示:

  ```
  db2-xdb:column="&quot;col one&quot;"
  ```

  （请注意，这些条件是特定于此注释的需求。）
- 在此注释中只能指定下列数据类型的列：CREATE TABLE SQL 语句支持的所有数据类型，但用户定义的结构类型除外。

详细信息

db2-xdb:column 注释是作为 XML 元素或属性声明中的属性或者作为 <db2-xdb:rowSetMapping> 的子元素指定的，它将 XML 元素或属性映射至目标表中的列名。使用此注释时，还必须指定 db2-xdb:rowSet 注释。它们共同描述用于存储此元素或属性的分解值的表和列。

示例

以下示例说明如何使用 db2-xdb:column 注释来将 <book> 元素内容插入到 BOOKCONTENTS 表的列中。下面首先给出带注释的模式的部分内容。

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer" />
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="ISBN" />
    <xs:attribute name="title" type="xs:string" />
  </xs:complexType>
</xs:element>

<xs:complexType name="chapterType">
  <xs:sequence>
    <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                db2-xdb:rowSet="BOOKCONTENTS"
                db2-xdb:column="CHPTCONTENT" />
  </xs:sequence>
  <xs:attribute name="number" type="xs:integer"
                db2-xdb:rowSet="BOOKCONTENTS"
                db2-xdb:column="CHPTNUM" />
  <xs:attribute name="title" type="xs:string"
                db2-xdb:rowSet="BOOKCONTENTS"
                db2-xdb:column="CHPTTITLE" />
</xs:complexType>
```

```
<xs:simpleType name="paragraphType">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

下面提供了所映射的 <book> 元素，然后是完成分解后得到的 BOOKCONTENTS 表。

```
<book isbn="1-11-111111-1" title="My First XML Book">
  <authorID>22</authorID>
  <!-- this book does not have a preface -->
  <chapter number="1" title="Introduction to XML">
    <paragraph>XML is fun...</paragraph>
    ...
  </chapter>
  <chapter number="2" title="XML and Databases">
    <paragraph>XML can be used with...</paragraph>
  </chapter>
  ...
  <chapter number="10" title="Further Reading">
    <paragraph>Recommended tutorials...</paragraph>
  </chapter>
</book>
```

表 58. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	Introduction to XML	XML is fun...
1-11-111111-1	2	XML and Databases	XML can be used with...
...	...	...	...
1-11-111111-1	10	Further Reading	Recommended tutorials...

db2-xdb:locationPath 分解注释

对于以全局方式声明的或者作为可复用组一部分声明的 XML 元素或属性，db2-xdb:locationPath 注释根据其祖代将其映射至不同的表/列对。可复用组是以全局方式声明的命名复杂类型、命名模型组和命名属性组。

db2-xdb:locationPath 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

<xs:element> 或 <xs:attribute> 的属性，或者 <db2-xdb:rowSetMapping> 的属性。

如何指定

通过下列任何一种方法来指定 db2-xdb:locationPath（其中 *value* 表示有效注释值）:

- `<xs:element db2-xdb:locationPath="value" />`
- `<xs:attribute db2-xdb:locationPath="value" />`
-

```
<db2-xdb:rowSetMapping db2-xdb:locationPath="value">
  <db2-xdb:rowSet>value</db2-xdb:rowSet>
  ...
</db2-xdb:rowSetMapping>
```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效值

db2-xdb:locationPath 的值必须使用以下语法:

```
location path := '/' (locationstep '/')* lastlocationstep
locationstep := (prefix':')? name
lastlocationstep := locationstep | '@' (prefix:)? name
```

其中，name 是元素名称或属性名，prefix 是名称空间前缀。

注意:

- 位置路径中使用的所有名称空间前缀都必须已经与模式文档中的名称空间相关联，并且该模式文档必须包含指定了此位置路径的注释。
- 可以通过对模式文档的 <xs:schema> 元素添加名称空间声明来创建名称空间前缀绑定。
- 如果 prefix 为空，那么认为 name 不在任何名称空间中。如果在模式文档中声明了缺省名称空间，并且 locationstep 中的名称属于该名称空间，那么必须声明缺省名称空间的名称空间前缀，并且必须使用此名称空间前缀来限定名称；在 db2-xdb:locationPath 中，空前缀并不表示缺省名称空间。

详细信息

db2-xdb:locationPath 注释用来描述以全局方式声明的或者作为下列任何一项的一部分声明的元素或属性的映射:

- 命名模型组
- 命名属性组
- 全局复杂类型声明
- 简单类型或复杂类型的全局元素或属性

对于无法复用的元素或属性声明（未包含在命名复杂类型定义、命名模型组或命名属性组中的本地声明）来说，db2-xdb:locationPath 注释不起作用。

当在各种祖代行中使用全局元素或属性声明作为引用（例如 <xs:element ref="abc">）时，应该使用 db2-xdb:locationPath。由于无法直接在引用中指定注释，所以，必须在相应全局元素或属性声明中指定那些注释。由于相应的元素或属性声明是全局的，所以，可以从 XML 模式中的许多不同上下文中引用该元素或属性。通常，应该使用 db2-xdb:locationPath 来区分这些不同上下文中的映射。对于命名复杂类型、命名模型组和命名属性组来说，应该在元素和属性的每个映射上下文中对它们的声明添加注释，以便进行分解。应该使用 db2-xdb:locationPath 注释来指定每个 locationPath 的目标 rowSet – column 对。同一个 db2-xdb:locationPath 值可以用于不同的 rowSet – column 对。

示例

以下示例说明如何根据属性所在的上下文来将同一个属性映射至不同的表。下面首先给出带注释的模式的部分内容。

```
<!-- global attribute -->
<xs:attribute name="title" type="xs:string"
              db2-xdb:rowSet="BOOKS"
              db2-xdb:column="TITLE"
              db2-xdb:locationPath="/books/book/@title">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping db2-xdb:locationPath="/books/book/chapter/@title">
        <db2-xdb:rowSet>BOOKCONTENTS</db2-xdb:rowSet>
        <db2-xdb:column>CHPTTITLE</db2-xdb:column>
      </db2-xdb:rowSetMapping>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>

<xs:element name="books">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="book">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="authorID" type="xs:integer" />
            <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attribute name="isbn" type="xs:string" />
          <xs:attribute ref="title" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="chapterType">
  <xs:sequence>
    <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="number" type="xs:integer" />
  <xs:attribute ref="title" />
</xs:complexType>

<xs:simpleType name="paragraphType">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

请注意，只有一个名为“title”的属性声明，但在不同的上下文中有两个对此属性的引用。其中一个引用在 <book> 元素中，另一个引用在 <chapter> 元素中。需要将“title”属性值根据上下文分解到不同的表中。此带注释的模式指定：如果“title”值是书名，那么将其分解到 BOOKS 表中，如果它是章节名，那么将其分解到 BOOKCONTENTS 表中。

下面提供了所映射的 <books> 元素，然后是完成分解后得到的 BOOKS 表。

```
<books>
  <book isbn="1-11-111111-1" title="My First XML Book">
    <authorID>22</authorID>
    <!-- this book does not have a preface -->
    <chapter number="1" title="Introduction to XML">
      <paragraph>XML is fun...</paragraph>
      ...
    </chapter>
```

```
    <chapter number="2" title="XML and Databases">
      <paragraph>XML can be used with...</paragraph>
    </chapter>
    ...
    <chapter number="10" title="Further Reading">
      <paragraph>Recommended tutorials...</paragraph>
    </chapter>
  </book>
...
</books>
```

表 59. BOOKS

ISBN	TITLE	CONTENT
NULL	My First XML Book	NULL

表 60. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
NULL	NULL	Introduction to XML	NULL
NULL	NULL	XML and Databases	NULL
...	...	...	...
NULL	NULL	Further Reading	NULL

db2-xdb:expression 分解注释

db2-xdb:expression 注释指定一个定制表达式，将把该表达式的结果插入到此元素映射至的表中。

db2-xdb:expression 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

<xs:element> 或 <xs:attribute> 的属性，或者 <db2-xdb:rowSetMapping> 的可选子元素，仅对包含列映射的注释有效。

如何指定

通过下列任何一种方法来指定 db2-xdb:expression（其中 *value* 表示有效注释值）:

- `<xs:element db2-xdb:expression="value" db2-xdb:column="value" />`
- `<xs:attribute db2-xdb:expression="value" db2-xdb:column="value" />`
-

```
<db2-xdb:rowSetMapping>
  <db2-xdb:rowSet>value</db2-xdb:rowSet>
  <db2-xdb:column>value</db2-xdb:column>
  <db2-xdb:expression>value</db2-xdb:expression>
  ...
</db2-xdb:rowSetMapping>
```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效值

db2-xdb:expression 的值必须使用以下语法，此语法为 SQL 表达式的一个子集:

```
expression := function (arglist) | constant | $DECOMP_CONTENT | $DECOMP_ELEMENTID |
   $DECOMP_DOCUMENTID | (scalar-fullselect) | expression operator expression |
   (expression) | special-register | CAST (expression AS data-type) |
   XMLCAST (expression AS data-type) | XML-function

operator := + | - | * | / | CONCAT

arglist := expression | arglist, expression
```

详细信息

db2-xdb:expression 注释使您能够指定一个定制表达式，当使用 $DECOMP_CONTENT 时，将把该表达式应用于所分解的 XML 元素或属性的内容。然后，将把对此表达式进行求值所产生的结果插入到分解期间指定的列中。

当您想要插入常量值（例如元素的名称）或者文档中未包含的生成值时，此注释也非常有用。

必须使用有效的 SQL 表达式来指定 db2-xdb:expression，所求值表达式的类型必须能够静态确定并且与插入该值时要使用的目标列的类型兼容。支持以下 SQL 表达式子集；不支持下面未描述的任何其他 SQL 表达式，那些表达式在此注释上下文中的行为是未定义的。

function (arglist)
: 内置标量 SQL 函数或者用户定义的标量 SQL 函数。标量函数的自变量是独立的标量值。标量函数返回单个值（可能为空）。有关更多信息，请参阅有关函数的文档。

constant
: 字符串常量值或数字常数值（有时称为字面值）。有关更多信息，请参阅有关常量的文档。

$DECOMP_CONTENT
: 文档中映射的 XML 元素或属性的值，此值是根据 db2-xdb:contentHandling 注释设置构造的。有关更多信息，请参阅分解关键字文档。

$DECOMP_ELEMENTID
: 系统生成的整数标识，它在 XML 文档中唯一地标识此注释所描述的元素或属性。有关更多信息，请参阅分解关键字文档。

$DECOMP_DOCUMENTID
: xdbDecompXML 存储过程的 *documentid* 输入参数中指定的字符串值，此值标识要分解的 XML 文档。有关更多信息，请参阅分解关键字文档。

(scalar-fullselect)
: 括在圆括号中的全查询，它返回一行，该行由单个列值组成。如果该全查询未返回行，那么表达式的结果为空值。

expression operator expression
: 以上受支持值列表中定义的两个受支持表达式操作数的结果。请参阅有关表达式的文档以了解有关表达式运算的详细信息。

(expression)
括在圆括号中的表达式，它符合上面定义的受支持表达式列表。

special-register
受支持专用寄存器的名称。此设置将求值为当前服务器的专用寄存器值。请参阅专用寄存器的文档以了解完整的受支持专用寄存器列表。

CAST (expression AS data-type)
如果表达式不为空，那么将该表达式的类型转换为指定的 SQL 数据类型。如果该表达式为空，那么结果是所指定 SQL 数据类型的空值。将空值插入到列中时，表达式必须将空值的类型转换为兼容的列类型（例如，对于整数列，执行 CAST (NULL AS INTEGER)）。

XMLCAST (expression AS data-type)
如果表达式不为空，那么将该表达式的类型转换为指定的数据类型。表达式或目标数据类型必须为 XML 类型。如果该表达式为空，那么目标类型必须为 XML，而结果是空的 XML 值。

XML-function
任何受支持的 SQL/XML 函数。

示例

以下示例说明如何使用 db2-xdb:expression 注释来将 XML 文档中的值应用于用户定义的函数。然后，将 UDF 返回的结果插入到数据库中，而不是插入文档本身中的值。下面首先给出带注释的模式的部分内容。

```
  <xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string" />
      <xs:element name="lastname" type="xs:string" />
      <xs:element name="activeStatus" type="xs:boolean" />
      <xs:attribute name="ID" type="xs:integer"
                    db2-xdb:rowSet="AUTHORS" db2-xdb:column="NUMBOOKS"
                    db2-xdb:expression="AuthNumBooks (INTEGER ($DECOMP_CONTENT))" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

假定有一个接收整数参数（此参数表示作者的标识）的用户定义的函数 AuthNumBooks，该函数返回系统中该作者所著书籍的总数。

下面提供所映射的 <author> 元素。

```
<author ID="22">
  <firstname>Ann</firstname>
  <lastname>Brown</lastname>
  <activeStatus>1</activeStatus>
</author>
```

$DECOMP_CONTENT 将被替换为 ID 属性实例中的值“22”。由于 $DECOMP_CONTENT 始终被替换为字符类型，并且由于 AuthNumBooks UDF 接收整数参数，所以 db2-xdb:expression 注释必须将 $DECOMP_CONTENT 的类型转换为整数。假定该 UDF 对此作者（其标识为 22）返回整数 8，那么会将 8 插入到 AUTHORS 表的 NUMBOOKS 列中，如下所示。

表 61. AUTHORS

AUTHID	FIRSTNAME	SURNAME	ACTIVE	NUMBOOKS
NULL	NULL	NULL	NULL	8

db2-xdb:condition 分解注释

db2-xdb:condition 注释指定一个条件，该条件确定是否将行插入到表中。可以将满足该条件的行插入到表中（这取决于行集的其他条件，如果有）；不会将不满足该条件的行插入到表中。

db2-xdb:condition 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

<xs:element> 或 <xs:attribute> 的属性，或者 <db2-xdb:rowSetMapping> 的可选子元素。无论条件所属的注释是否包含列映射，都会应用该条件。

如何指定

通过下列任何一种方法来指定 db2-xdb:condition（其中 *value* 表示有效注释值）：

- `<xs:element db2-xdb:condition="value" />`
- `<xs:attribute db2-xdb:condition="value" />`
-
```
<db2-xdb:rowSetMapping>
  <db2-xdb:rowSet>value</db2-xdb:rowSet>
  <db2-xdb:condition>value</db2-xdb:condition>
  ...
</db2-xdb:rowSetMapping>
```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效值

下列类型的 SQL 谓词：基本谓词、量化谓词、BETWEEN、EXISTS、IN、IS VALIDATED、LIKE、NULL 和 XMLEXISTS。这些谓词还必须由 db2-xdb:expression 注释和/或列名支持的表达式组成。

详细信息

db2-xdb:condition 注释使您能够指定一些条件，在分解期间，将根据这些条件将值插入到数据库中。此注释通过应用用户指定的条件来对行进行过滤。在分解期间，将把满足指定条件的行插入到数据库中，而不会插入不满足条件的行。

如果同一个行集的多个元素或属性声明中指定了 db2-xdb:condition 注释，那么仅当所有条件的逻辑 AND 求值为 true 时才会插入该行。

db2-xdb:condition 中的列名

由于 db2-xdb:condition 由 SQL 谓词组成，所以可以在此注释中指定列名。如果涉及行集的 db2-xdb:condition 注释包含未限定的列名，那么在所有涉及该行集的映射中都必须存在对该列的映射。当在包含 SELECT 语句的谓词中使用其他列名时，必须对那些列名进行限定。如果 db2-xdb:condition 指定了未限定的列名，但未对指定了 db2-xdb:condition 的元素或属性指定列映射，那么，在对该条件进行求值时，所求的值是映射至所引用列名的元素或属性的内容。

请考虑以下示例:

```
<xs:element name="a" type="xs:string"
           db2-xdb:rowSet="rowSetA" db2-xdb:condition="columnX='abc'" />
<xs:element name="b" type="xs:string"
           db2-xdb:rowSet="rowSetB" db2-xdb:column="columnX" />
```

请注意，未对 <a> 指定列映射，但条件引用了列“columnX”。在对该条件进行求值时，将把该条件中的“columnX”替换为 <b> 中的值。这是因为，<b> 对“columnX”指定了列映射，而 <a> 没有列映射。如果 XML 文档包含:

```
<a>abc</a>
<b>def</b>
```

那么，由于在条件中对 <b> 的值“def”进行了求值，所以此例中的条件将求值为 false。

如果在与元素 <a> 的声明相关的 db2-xdb:condition 中使用了 $DECOMP_CONTENT（一个分解关键字，它以字符数据形式指定映射的元素或属性的值）而不是列名，那么将使用 <a>（而不是 <b>）的值来对条件进行求值。

```
<xs:element name="a" type="xs:string"
           db2-xdb:rowSet="rowSetA" db2-xdb:condition="$DECOMP_CONTENT='abc'" />
<xs:element name="b" type="xs:string"
           db2-xdb:rowSet="rowSetB" db2-xdb:column="columnX" />
```

如果 XML 文档包含:

```
<a>abc</a>
<b>def</b>
```

那么，在本例中，由于求值时使用了 <a> 的值“abc”，所以条件将求值为 true。

当您只想根据另一个不会插入到数据库中的元素或属性值来分解值时，这种条件处理方式（使用列名和 $DECOMP_CONTENT）就非常有用。

文档未包含对映射的元素或属性指定的条件

如果对元素或属性指定了条件，但 XML 文档未包含该元素或属性，在这种情况下，仍然会应用该条件。例如，考虑带注释的模式文档中的以下元素映射:

```
<xs:element name="intElem" type="xs:integer"
           db2-xdb:rowSet="rowSetA" db2-xdb:column="colInt"
           db2-xdb:condition="colInt > 100" default="0" />
```

即使 XML 文档未包含 <intElem> 元素，也仍然会对条件“colInt > 100”进行求值。由于 <intElem> 未出现，所以对“colInt”进行条件求值时将使用缺省值 0。于是，该条件是作为 0 > 100 求值的，这将求值为 false。因此，分解期间不会插入相应的行。

示例

考虑 XML 文档中的以下 <author> 元素:

```
<author ID="0800">
    <firstname>Alexander</firstname>
    <lastname>Smith</lastname>
      <activeStatus>1</activeStatus>
  </author>
```

将根据 db2-xdb:condition 指定的条件，来确定是否会在分解期间将此 <author> 元素中的值插入到目标表中。下面提供了两种情况。

所有条件都满足

在与上面的 <author> 元素相对应的带注释的模式中，下面这部分内容指定仅当作者的标识是 1 到 999 之间、<firstname> 和 <lastname> 元素不为空并且 <activeStatus> 元素值等于 1 时，才应该对此元素进行分解:

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="GIVENNAME"
                  db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL" />
      <xs:element name="lastname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="SURNAME"
                  db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL" />
      <xs:element name="activeStatus" type="xs:integer"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="statusCode"
                  db2-xdb:condition="$DECOMP_CONTENT=1" />
      <xs:attribute name="ID" type="xs:integer"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="AUTHID"
                  db2-xdb:condition="$DECOMP_CONTENT BETWEEN 1 and 999 />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

由于以上 <author> 元素示例中的值满足 db2-xdb:condition 指定的所有条件，所以，<author> 元素中的数据将填充到 AUTHORS 表中。

表 62. AUTHORS

AUTHID	GIVENNAME	SURNAME	STATUSCODE	NUMBOOKS
0800	Alexander	Smith	1	NULL

一个条件失败

以下带注释的模式指定: 仅当作者的标识是 1 到 100 并且 <firstname> 和 <lastname> 元素不为空时，才应该分解 <author> 元素:

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="GIVENNAME"
                  db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL"/>
      <xs:element name="lastname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="SURNAME"
                  db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL"/>
      <xs:element name="activeStatus" type="xs:integer" />
      <xs:attribute name="ID" type="xs:integer"
                    db2-xdb:rowSet="AUTHORS" db2-xdb:column="AUTHID"
```

```
                   db2-xdb:condition="$DECOMP_CONTENT BETWEEN 1 and 100 />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

虽然 <author> 元素示例的 <firstname> 和 <lastname> 元素符合指定的条件，但由于 ID 属性值不符合条件，所以在分解期间整行都不会被插入。这是因为对 AUTHORS 表上指定的三个条件的逻辑 AND 都进行了求值。在本例中，其中一个条件为 false，因此逻辑 AND 求值为 false，所以未插入任何行。

db2-xdb:contentHandling 分解注释

db2-xdb:contentHandling 注释指定将分解到复杂类型或简单类型元素表的内容类型。

db2-xdb:contentHandling 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

<xs:element> 的属性，或者应用于复杂类型或简单类型元素声明的 <db2-xdb:rowSetMapping> 的属性

如何指定

通过下列任何一种方法来指定 db2-xdb:contentHandling（其中 *value* 表示有效注释值）：

- `<xs:element db2-xdb:contentHandling="value" />`
- ```
  <db2-xdb:rowSetMapping db2-xdb:contentHandling="value">
    <db2-xdb:rowSet>value</db2-xdb:rowSet>
    ...
  </db2-xdb:rowSetMapping>
  ```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效值

下列其中一个区分大小写的标记:

- text
- stringValue
- serializeSubtree

详细信息

db2-xdb:contentHandling 注释是作为 XML 元素声明的属性指定的，它指示在分解期间要插入到分别由 db2-xdb:rowSet 和 db2-xdb:column 指定的表和列中的值。db2-xdb:contentHandling 的三个有效值是:

text

- 插入的内容：此元素中字符数据（包括 CDATA 部分的字符内容）的并置。
- 排除的内容：此元素的注释和处理指令、CDATA 部分定界符（“<![CDATA[″ ″]]>”）以及此元素的后代（包括标记和内容）。
  ```

stringValue

- 插入的内容：此元素中的字符数据（包括 CDATA 部分的字符内容）与此元素后代中的字符数据的并置（按文档顺序并置）。
- 排除的内容：注释、处理指令、CDATA 部分定界符（“<![CDATA[″ ″]]>”）以及此元素的后代的起始和结束标记。

serializeSubtree

- 插入的内容：此元素的起始标记与结束标记之间所有内容的标记，包括此元素的起始标记和结束标记。这些内容包括注释、处理指令和 CDATA 部分定界符（“<![CDATA[″ ″]]>”）。
- 排除的内容：无。
- 注意：插入的序列化字符串可能与 XML 文档中的相应部分并不完全相同，这可能是由下列因素引起的：XML 模式中指定的缺省值、实体扩展、属性顺序、属性的空格规范化以及对 CDATA 部分进行的处理。

  由于此设置产生的序列化字符串是 XML 实体，所以应该考虑代码页问题。如果目标列是字符类型或图形类型，那么将使用数据库代码页来插入 XML 片段。由于 XML 处理器无法自动检测除 UTF-8 以外的编码，所以当应用程序将此类实体传递给XML 处理器时，该应用程序必须显式地将该实体的编码告知该处理器。但是，如果目标列是 BLOB 类型，就会使用 UTF-8 编码来插入 XML 实体。在这种情况下，不需要指定编码就可以将 XML 实体传递给 XML 处理器。

如果进行注释以便分解的 XML 元素声明是复杂类型并且包含复杂内容，但未指定 db2-xdb:contentHandling，那么缺省行为遵循“serializeSubtree”设置。对于带注释元素声明的所有其他情况，如果未指定 db2-xdb:contentHandling，那么缺省行为遵循“stringValue”设置。

如果将元素声明为复杂类型并且具有仅元素或空内容模型（即，元素声明的“mixed”属性未设置为 true 或 1），那么不能将 db2-xdb:contentHandling 设置为“text”。

对元素指定 db2-xdb:contentHandling 注释并不会影响该元素任何后代的分解。

db2-xdb:contentHandling 的设置会影响替换 db2-xdb:expression 或 db2-xdb:condition 注释中的 $DECOMP_CONTENT 的值。首先根据 db2-xdb:contentHandling 设置对替换的值进行处理，然后才会传递该值以进行求值。

请注意，如果已在分解之前或者分解期间执行了验证，那么 db2-xdb:contentHandling 所处理内容的实体已进行了解析。

示例

以下示例说明如何使用不同的 db2-xdb:contentHandling 注释设置来在目标表中获得不同的结果。首先给出带注释的模式，它说明如何使用 db2-xdb:contentHandling 来对 <paragraph> 元素添加注释。（带注释的模式只提供一次，它将 db2-xdb:contentHandling 设置为“text”。本节中的后续示例使用同一个带注释的模式，而仅仅是设置的 db2-xdb:contentHandling 值有所不同。）

```
<xs:schema>
  <xs:element name="books">
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element name="book">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="authorID" type="xs:integer" />
              <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
            </xs:sequence>
            <xs:attribute name="isbn" type="xs:string"
                          db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="ISBN" />
            <xs:attribute name="title" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="chapterType">
    <xs:sequence>
      <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTCONTENT"
                  db2-xdb:contentHandling="text" />
    </xs:sequence>
    <xs:attribute name="number" type="xs:integer"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTNUM" />
    <xs:attribute name="title" type="xs:string"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTTITLE" />
  </xs:complexType>

  <xs:complexType name="paragraphType" mixed="1">
    <xs:choice>
      <xs:element name="b" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
    </xs:choice>
  </xs:complexType>
</xs:schema>
```

下面提供所映射的 <books> 元素。

```
<books>
  <book isbn="1-11-111111-1" title="My First XML Book">
    <authorID>22</authorID>
    <chapter number="1" title="Introduction to XML">
      <paragraph>XML is <b>lots of</b> fun...</paragraph>
    </chapter>
    <chapter number="2" title="XML and Databases">
      <paragraph><!-- Start of chapter -->XML can be used with...</paragraph>
      <paragraph><?processInstr example?>
        Escape characters such as <![CDATA[ <, >, and & ]]>...</paragraph>
    </chapter>
    ...
    <chapter number="10" title="Further Reading">
      <paragraph>Recommended tutorials...</paragraph>
    </chapter>
  </book>
  ...
<books>
```

下面三个表显示使用不同的 db2-xdb:contentHandling 值对同一个 XML 元素进行分解所产生的结果。

注: 在以下结果表的 CHPTTITLE 和 CHPTCONTENT 列中，值引在引号中。这些引号在列中不存在，这里提供这些引号的目的仅仅是为了指示插入的字符串的边界和空格。

db2-xdb:contentHandling=″text″

表 63. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	″Introduction to XML″	″XML is fun...″
1-11-111111-1	2	″XML and Databases″	″XML can be used with...″
1-11-111111-1	2	″XML and Databases″	″ Escape characters such as <, >, and & ...″
...	...	...	...
1-11-111111-1	10	″Further Reading″	″Recommended tutorials...″

请注意，使用“text”设置时，未插入第一章第一段中 元素的内容。这是因为“text”设置将排除后代中的任何内容。另请注意，使用“text”设置时，排除了第二章第一段中的注释和处理指令。保留了 <paragraph> 元素中字符数据并置中的空格。

db2-xdb:contentHandling=″stringValue″

表 64. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	″Introduction to XML″	″XML is lots of fun...″
1-11-111111-1	2	″XML and Databases″	″XML can be used with...″
1-11-111111-1	2	″XML and Databases″	″ Escape characters such as <, >, and & ...″
...	...	...	...
1-11-111111-1	10	″Further Reading″	″Recommended tutorials...″

此表与上一个表之间的区别是第一行的 CHPTCONTENT 列有所不同。请注意，插入了字符串“lots of”，此字符串来自 <paragraph> 元素的 后代。当 db2-xdb:contentHandling 设置为“text”时，由于“text”设置排除后代的内容，所以排除了此字符串。但是，“stringValue”设置包括后代的内容。与“text”设置相同，未插入注释和处理指令，并保留了空格。

db2-xdb:contentHandling=″serializeSubtree″

表 65. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	″Introduction to XML″	″<paragraph>XML is lots of fun...</paragraph>″

表 65. BOOKCONTENTS (续)

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	2	″XML and Databases″	″<paragraph><!-- Start of chapter -->XML can be used with...</paragraph>″
1-11-111111-1	2	″XML and Databases″	″<paragraph><?processInstr example?> Escape characters such as <![CDATA[<, >, and &]]>...</paragraph>″
...	...	...	...
1-11-111111-1	10	″Further Reading″	″<paragraph>Recommended tutorials...</paragraph>″

此表与前两个表的区别是，插入了 <paragraph> 元素的后代中的所有标记（包括 <paragraph> 的起始和结束标记）。这包括第一行 CHPTCONTENT 列中的 <b> 起始和结束标记，以及第二行和第三行中分别包含的注释和处理指令。与上两个示例相同，保留了 XML 文档中的空格。

db2-xdb:normalization 分解注释

db2-xdb:normalization 注释指定要插入或替换 $DECOMP_CONTENT（与 db2-xdb:expression 配合使用时）的 XML 数据中的空格的规范化。

db2-xdb:normalization 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

<xs:element> 或 <xs:attribute> 的属性，或者 <db2-xdb:rowSetMapping> 的属性。

如何指定

通过下列任何一种方法来指定 db2-xdb:normalization（其中 *value* 表示有效注释值）：

- `<xs:element db2-xdb:normalization="value" />`
- `<xs:attribute db2-xdb:normalization="value" />`
-
```
<db2-xdb:rowSetMapping db2-xdb:normalization="value">
  <db2-xdb:rowSet>value</db2-xdb:rowSet>
  ...
</db2-xdb:rowSetMapping>
```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效值

下列其中一个区分大小写的标记:

- canonical
- original（缺省值）

- whitespaceStrip

注: db2-xdb:normalization 属性仅对某些 XML 模式类型与 SQL 字符类型之间的映射有效。请参阅"详细信息"一节以了解对于 SQL 字符列来说可以规范化的受支持 XML 模式类型列表。

详细信息

将 XML 值插入到字符类型目标列（CHAR、VARCHAR、LONG VARCHAR、CLOB、DBCLOB、GRAPHIC、VARGRAPHIC 和 LONG VARGRAPHIC）时，可能需要对所插入的数据进行规范化。可以使用 db2-xdb:normalization 注释来指定不同类型的规范化；有效值（区分大小写的设置）如下所示:

canonical
: 将 XML 值插入到目标列中或者用于替换此 db2-xdb:normalization 注释所在映射中出现的 $DECOMP_CONTENT 之前，根据该值的 XML 模式类型将其转换为规范格式。

original
: 在 XML 解析器对元素内容或属性值（取决于此映射是用于 XML 元素还是 XML 属性）的原始字符数据进行任何处理后，将该字符数据插入到目标列中或者用于替换此 db2-xdb:normalization 注释所在映射中出现的 $DECOMP_CONTENT。如果未对与此注释相关的映射指定 db2-xdb:normalization 属性，分解过程就会根据"original"设置将数据规范化。

whitespaceStrip
: 将 XML 值插入到目标列中或者替换此 db2-xdb:normalization 注释所在映射中出现的 $DECOMP_CONTENT 之前，除去该 XML 值中的所有前导和尾部空格，并将连续的空格折叠成单个空格字符。

在将下列其中一种原子 XML 模式类型（或者派生自这些原子 XML 模式类型）的元素或属性映射至字符类型（CHAR、VARCHAR、LONG VARCHAR、CLOB、DBCLOB、GRAPHIC、VARGRAPHIC 和 LONG VARGRAPHIC）的列时，db2-xdb:normalization 适用。

- byte 和 unsigned byte
- integer、positiveInteger、 negativeInteger、 nonPositiveInteger 和 nonNegativeInteger
- int 和 unsignedInt
- long 和 unsignedLong
- short 和 unsignedShort
- decimal
- float
- double
- boolean
- time
- date
- dateTime

如果对任何其他类型指定 db2-xdb:normalization，它将被忽略。请注意，这些就是在 W3C 建议 XML Schema Part 2: Datatypes Second Edition 中指定了规范表示的 XML 模式类型。

由于 db2-xdb:normalization 注释仅对某些 XML 模式到 SQL 字符类型的映射有效，因此，如果对不受支持的映射指定此注释，它就会被忽略。

示例

以下示例说明如何使用 db2-xdb:normalization 注释来控制空格规范化。首先给出带注释的模式。

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="FIRSTNAME" />
      <xs:element name="lastname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="SURNAME"
                  db2-xdb:normalization="whitespaceStrip" />
      <xs:element name="activeStatus" type="xs:boolean"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="ACTIVE"
                  db2-xdb:normalization="canonical" />
      <xs:attribute name="ID" type="xs:integer"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="AUTHID"
                  db2-xdb:normalization="whitespaceStrip" />
</xs:sequence>
  </xs:complexType>
</xs:element>
```

下面提供了所映射的 <author> 元素（在以下内容中，为了便于说明，将值得注意的空格表示成“_”下划线字符），然后是完成分解后得到的 AUTHORS 表。

```
<author ID="__22">
   <firstname>Ann</firstname>
   <lastname>__Brown_</lastname>
   <activeStatus>1</activeStatus>
 </author>
```

表 66. AUTHORS

AUTHID	FIRSTNAME	SURNAME	ACTIVE	NUMBOOKS
22	Ann	__Brown_	true	NULL

“whitespaceStrip”设置导致在将值插入到目标表中之前从“ID”属性中除去前导空格。但是，请注意，即使指定了“whitespaceStrip”设置，也不会从 <lastname> 元素中除去前导和尾部空格。这是因为，<lastname> 元素的 XML 模式类型为字符串，db2-xdb:normalization 不适用于该类型。<author> 的 <activeStatus> 子元素被定义为布尔类型，布尔类型的规范表示是字面值“true”或“false”。<activeStatus> 元素的“canonical”设置导致将规范格式“1”（即“true”）插入到 AUTHORS 表的 ACTIVE 列中。

在上面给出的 XML 模式中，如果已使用 db2-xdb:normalization=″original″ 对“ID”属性添加了注释，那么将把文档中的原始值“__22”（下划线字符表示空格）插入到 AUTHID 列中。

db2-xdb:order 分解注释

db2-xdb:order 注释指定不同表之间的行插入顺序。

db2-xdb:order 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

<db2-xdb:rowSetOperationOrder> 的子元素

如何指定

通过以下方法来指定 db2-xdb:order（其中 *value* 表示有效注释值）：

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetOperationOrder>
        <db2-xdb:order>
          <db2-xdb:rowSet>value</db2-xdb:rowSet>
          ...
        </db2-xdb:order>
      </db2-xdb:rowSetOperationOrder>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效结构

下面是受支持的 <db2-xdb:order> 子元素:

db2-xdb:rowSet
: 指定映射至目标基本表的 XML 元素或属性。

详细信息

db2-xdb:order 注释用于定义属于给定行集的行的插入顺序（相对于属于另一个行集的行）。这允许将 XML 数据插入到目标表中，其方式符合对关系模式中的表定义的所有引用完整性约束。

如果在 db2-xdb:order 中 RS1 列示在 RS2 之前，那么会在属于行集 RS2 的所有行之前插入给定行集 RS1 的所有行。可指定此元素的多个实例以定义多个插入顺序层次结构。对于未出现在任何元素中的行集，它们的行可按任意顺序插入（相对于任何其他行集的行）。而且每个 <db2-xdb:rowSet> 元素的内容必须是显式定义的行集，或未对其创建任何显式行集声明的现有表。

可定义多个行集插入层次结构，尽管行集只能出现在 <db2-xdb:order> 元素的一个实例中，而且只能在该元素中出现一次。

对于子元素中指定的定界 SQL 标识来说，必须将引号定界符包括在字符内容中，并且不必进行转义。但是，必须对 SQL 标识中使用的“&”和“<”字符进行转义。

示例

以下示例说明如何使用 db2-xdb:order 注释。

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetOperationOrder>

        <db2-xdb:order>
          <db2-xdb:rowSet>CUSTOMER</db2-xdb:rowSet>
          <db2-xdb:rowSet>PURCHASE_ORDER</db2-xdb:rowSet>
        </db2-xdb:order>

        <db2-xdb:order>
          <db2-xdb:rowSet>ITEMS_MASTER</db2-xdb:rowSet>
          <db2-xdb:rowSet>PO_ITEMS</db2-xdb:rowSet>
        </db2-xdb:order>

      </db2-xdb:rowSetOperationOrder>
    </xs:appinfo>
  </xs:annotation>
</xs:schema>
```

上述示例中指定了插入顺序的两个不相交层次结构。第一个层次结构指定 CUSTOMER 行集或表的所有内容在为 PURCHASE_ORDER 收集的所有内容之前插入，而第二个层次结构指定 ITEMS_MASTER 行集的所有内容在向 PO_ITEMS 中插入任何内容之前插入。注意，未定义两个层次结构之间的顺序。例如，PURCHASE_ORDER 行集或表的所有内容可在向 ITEMS_MASTER 中插入任何内容之前或之后插入。

限制

指定行集插入顺序存在下列限制:

- 在 32 位系统上，分解带有插入顺序要求的大型文档可能导致系统内存不足。
- 在 64 位系统上，如果管理员限制了进程允许的虚拟内存空间，那么可能会发生内存不足的情况。对进程指定足够大或不受限制的虚拟内存设置有助于避免内存不足的情况，但这会对系统的整体性能带来负面影响。

db2-xdb:truncate 分解注释

db2-xdb:truncate 注释指定在将 XML 值插入字符目标列时是否允许进行截断。

db2-xdb:truncate 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

<xs:element> 或 <xs:attribute> 的属性，或者 <db2-xdb:rowSetMapping> 的属性。

如何指定

通过下列任何一种方法来指定 db2-xdb:truncate（其中 *value* 表示有效注释值）:

- `<xs:element db2-xdb:truncate="value" />`
- `<xs:attribute db2-xdb:truncate="value" />`

```
• <db2-xdb:rowSetMapping db2-xdb:truncate="value">
    <db2-xdb:rowSet>value</db2-xdb:rowSet>
    ...
  </db2-xdb:rowSetMapping>
```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效值

下列其中一个标记:

- 0（等于 false；缺省值）
- 1（等于 true）
- false（区分大小写；缺省值）
- true（区分大小写）

详细信息

插入到目标字符列中的 XML 值可能大于列大小，在这种情况下，必须截断该值才能成功地进行分解。db2-xdb:truncate 属性指示当值对于目标列来说太大时是否允许截断该值。如果此属性设置为“false”或“0”（表示不允许截断值），并且插入的 XML 值对于目标列来说太大，在分解 XML 文档期间就会出错，并且不会插入该值。“true”或“1”设置表示允许在插入期间截断数据。

db2-xdb:truncate 仅在目标列符合下列条件时适用:

- 目标列具有字符类型，或者
- 目标列具有 DATE、TIME 或 TIMESTAMP 类型，并且 XML 值分别具有 xs:date、xs:time 或 xs:dateTime 类型。

如果在 db2-xdb:truncate 所在的元素或属性声明中指定了 db2-xdb:expression 注释，就会忽略 db2-xdb:truncate 值，这是因为，如果将表达式定义为可以执行截断，就会执行此项操作。

在将指定了时区并且 XML 模式类型为日期、时间或时间戳记的 XML 值分解到 SQL 日期时间列中时，必须将 db2-xdb:truncate 设置为“true”或“1”。这是因为 SQL 日期时间类型的结构不允许指定时区。

示例

以下示例说明如何对 <author> 元素应用截断功能。下面首先给出带注释的模式的部分内容。

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="FIRSTNAME"
                  db2-xdb:truncate="true" />
      <xs:element name="lastname" type="xs:string" />
      <xs:element name="activeStatus" type="xs:boolean" />
      <xs:element name="activated" type="xs:date"
                  db2-xdb:truncate="true" />
```

```
      <xs:attribute name="ID" type="xs:integer" />
    <xs:sequence>
  </xs:complexType>
  </xs:element>
```

下面提供所映射的 <author> 元素。

```
<author ID="0800">
  <firstname>Alexander</firstname>
  <lastname>Smith</lastname>
  <activeStatus>0</activeStatus>
  <activated>2001-10-31Z</activated>
  </author>
```

假定 FIRSTNAME 列定义为 CHAR SQL 类型并且大小为 7，并且 ACTIVEDATE 列定义为 DATE SQL 类型。下面提供了完成分解后得到的 AUTHORS 表。

表 67. AUTHORS

AUTHID	FIRSTNAME	SURNAME	ACTIVE	ACTIVEDATE	NUMBOOKS
NULL	Alexand	NULL	NULL	2001-10-31	NULL

由于 <firstname> 值“Alexander”的长度大于 SQL 列大小，所以需要执行截断才能插入该值。另请注意，由于 XML 文档中的 <activated> 元素包含时区，所以已将 db2-xdb:truncate 设置为“true”以确保分解期间能够成功地插入该日期。

由于需要执行截断才能插入 <firstname> 元素值或 <activated> 元素值，所以，如果未指定 db2-xdb:truncate，就会采用 db2-xdb:truncate 的缺省值（不允许执行截断），并且将生成错误以指示未插入值。

db2-xdb:rowSetMapping 分解注释

<db2-xdb:rowSetMapping> 注释将单个 XML 元素或属性映射至一个或多个表/列对。

<db2-xdb:rowSetMapping> 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

作为 <xs:element> 或 <xs:attribute> 的子元素的 <xs:appinfo>（这是 <xs:annotation> 的子元素）的子元素

如何指定

通过下列任何一种方法来指定 db2-xdb:rowSetMapping（其中 *value* 表示有效注释值）：

- ```
  <xs:element>
    <xs:annotation>
      <xs:appinfo>
        <db2-xdb:rowSetMapping>
          <db2-xdb:rowSet>value</db2-xdb:rowSet>
          ...
        </db2-xdb:rowSetMapping>
      </xs:appinfo>
    </xs:annotation>
    ...
  </xs:element>
  ```
  ```

- ```
  <xs:attribute>
    <xs:annotation>
      <xs:appinfo>
        <db2-xdb:rowSetMapping>
          <db2-xdb:rowSet>value</db2-xdb:rowSet>
          ...
        </db2-xdb:rowSetMapping>
      </xs:appinfo>
    </xs:annotation>
    ...
  </xs:attribute>
  ```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效结构

受支持的 <db2-xdb:rowSetMapping> 属性如下所示:

db2-xdb:contentHandling
: 允许指定将分解到复杂类型元素表的元素内容。

db2-xdb:locationPath
: 允许将作为可复用组一部分声明的 XML 元素或属性映射至不同的表/列对（此映射是根据该元素或属性的祖代进行的）。

db2-xdb:normalization
: 允许指定插入内容前对映射至字符目标列的 XML 元素或属性内容执行的规范化行为。

db2-xdb:truncate
: 允许指定在将 XML 值插入字符目标列时是否允许进行截断。

<db2-xdb:rowSetMapping> 的这些属性也是 XML 元素或属性声明的属性；无论它们是 <db2-xdb:rowSetMapping> 的属性还是 <xs:element> 或 <xs:attribute> 的属性，行为和需求都是相同的。请参阅这些注释的各个相应文档以了解详细信息。

受支持的 <db2-xdb:rowSetMapping> 子元素如下所示，按照指定它们时必须遵循的顺序进行列示:

<db2-xdb:rowSet>
: 将 XML 元素或属性映射至目标基本表。

<db2-xdb:column>
: （可选）将 XML 元素或属性映射至基本表列。如果 db2-xdb:rowSetMapping 注释包含 db2-xdb:expression，那么此元素是必需的。

  如果未计划向表插入值，那么 <db2-xdb:column> 是可选的，但只能用于条件处理。例如，如果要根据第二个元素的值来分解第一个元素，那么由于不会插入第二个元素的值，所以第二个元素不需要列映射。

<db2-xdb:expression>
: （可选）指定定制表达式，将把该表达式的结果插入到 db2-xdb:rowSet 属性指定的表中。
  ```

如果 db2-xdb:expression 指定了 $DECOMP_CONTENT，并且在同一映射中指定了 db2-xdb:normalization，那么在合适的情况下，在将 db2-xdb:expression 的 $DECOMP_CONTENT 值传递给该表达式以进行求值之前，将对该值进行规范化。

<db2-xdb:condition>

（可选）指定求值条件。

请注意，<db2-xdb:rowSetMapping> 的这些子元素与它们的相应属性注释具有相同的语义和语法，只是不需要对引号进行转义。

要了解更多详细信息，请参阅这些注释的属性版本的相应文档。

详细信息

可以使用 <db2-xdb:rowSetMapping> 来将 XML 元素或属性映射至单个目标表和列、映射至同一个表的多个目标列或者映射至多个表和列。可以通过两种等同的方法来映射至单个表和列：组合使用 db2-xdb:rowSet 和 db2-xdb:column 注释（它们是所映射的元素或属性的属性），或者指定 <db2-xdb:rowSetMapping>（它是所映射的元素或属性的子元素）。这两种方法能获得相同的结果，它们仅仅是表示法有所不同。

在 <db2-xdb:rowSetMapping> 的子元素的字符内容中，所有空格都是有意义的；不会执行空格规范化。对于子元素中指定的定界 SQL 标识来说，必须将引号定界符包括在字符内容中，并且不能对其进行转义。但是，必须对 SQL 标识中使用的“&”和“<”字符进行转义。

示例

以下示例说明如何使用 <db2-xdb:rowSetMapping> 注释来将名为“isbn”的单个属性映射至多个表。下面首先给出带注释的模式的部分内容。此部分内容说明如何将 isbn 值映射至 BOOKS 和 BOOKCONTENTS 表。

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer"/>
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string">
      <xs:annotation>
        <xs:appinfo>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>BOOKS</db2-xdb:rowSet>
            <db2-xdb:column>ISBN</db2-xdb:column>
          </db2-xdb:rowSetMapping>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>BOOKCONTENTS</db2-xdb:rowSet>
            <db2-xdb:column>ISBN</db2-xdb:column>
          </db2-xdb:rowSetMapping>
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="title" type="xs:string" />
  </xs:complexType>
</xs:element>
```

下面提供了所映射的 <book> 元素，然后是完成分解后得到的 BOOKS 和 BOOKCONTENTS 表。

```
<book isbn="1-11-111111-1" title="My First XML Book">
  <authorID>22</authorID>
  <!-- this book does not have a preface -->
  <chapter number="1" title="Introduction to XML">
    <paragraph>XML is fun...</paragraph>
    ...
  </chapter>
  ...
</book>
```

表 68. *BOOKS*

ISBN	TITLE	CONTENT
1-11-111111-1	NULL	NULL

表 69. *BOOKCONTENTS*

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	NULL	NULL	NULL

使用 <db2-xdb:rowSetMapping> 、db2-xdb:rowSet 及 db2-xdb:column 的组合的备用映射

以下带注释的模式部分等同于上面提供的 XML 模式片段，它们产生相同的分解结果。这两种模式之间的区别是，此模式将一个映射替换为 db2-xdb:rowSet 与 db2-xdb:column 的组合，而不是只使用 <db2-xdb:rowSetMapping> 注释。

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer"/>
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string"
              db2-xdb:rowSet="BOOKS" db2-xdb:column="ISBN" >
      <xs:annotation>
        <xs:appinfo>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>BOOKCONTENTS</db2-xdb:rowSet>
            <db2-xdb:column>ISBN</db2-xdb:column>
          </db2-xdb:rowSetMapping>
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="title" type="xs:string" />
  </xs:complexType>
</xs:element>
```

db2-xdb:rowSetOperationOrder 分解注释

db2-xdb:rowSetOperationOrder 注释是一个或多个 db2-xdb:order 元素的父代。请参阅有关 db2-xdb:order 的一节以了解有关定义不同表间的行插入顺序的用法的详细信息。

db2-xdb:rowSetOperationOrder 属于可添加到 XML 模式文档中的分解注释集，使用它来描述 XML 文档元素和属性与 DB2 基本表之间的映射。分解过程使用带注释的 XML 模式来确定应该如何将 XML 文档元素和属性分解到 DB2 表中。

注释类型

作为全局 <xs:annotation> 元素子代的 <xs:appinfo> 的子元素。

如何指定

通过以下方法来指定 db2-xdb:rowSetOperationOrder（其中 *value* 表示有效注释值）:

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetOperationOrder>
        <db2-xdb:order>
          <db2-xdb:rowSet>value</db2-xdb:rowSet>
          ...
        </db2-xdb:order>
      </db2-xdb:rowSetOperationOrder>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

名称空间

http://www.ibm.com/xmlns/prod/db2/xdb1

有效结构

下面是受支持的 <db2-xdb:rowSetOperationOrder > 子元素:

db2-xdb:order

详细信息

<db2-xdb:rowSetOperationOrder> 会将 <db2-xdb:order> 元素组合到一起。子 <db2-xdb:order> 元素可存在多个实例，以允许定义多个插入层次结构。

通过允许控制插入 XML 文档内容的顺序，db2-xdb:rowSetOperationOrder 和 db2-xdb:order 注释一起提供了一种方法，以确保 XML 模式分解进程尊重针对目标表的所有引用完整性约束以及规定在插入一个表的若干行之前插入另一个表的若干行的任何其他应用程序要求。

db2-xdb:rowSetOperationOrder 注释只能在 XML 模式中出现一次。

示例

请参阅有关 db2-xdb:order 注释的一节，以获取指定行集插入的示例。

带注释的 XML 模式分解的关键字

带注释的 XML 模式分解提供了分解关键字来供 db2-xdb:condition 和 db2-xdb:expression 注释使用。

$DECOMP_CONTENT

文档中映射的 XML 元素或属性的值，此值是根据 db2-xdb:contentHandling 注释设置构造的。应该始终将表达式中 $DECOMP_CONTENT 的替换值视为字符类型。请参阅局限性与限制文档以了解所支持的 $DECOMP_CONTENT 实例的最大字符串长度和实例数。如果 db2-xdb:expression 指定了 $DECOMP_CONTENT，并且在同一映射中指定了 db2-xdb:normalization，那么在合适的情况下，在将 db2-xdb:expression 的 $DECOMP_CONTENT 值传递给该表达式以进行求值之前，将对该值进行规范化。

可使用 $DECOMP_CONTENT 来处理所映射元素或属性的值（使用定制表达式来进行处理，而不是直接插入该值）。

$DECOMP_DOCUMENTID

xdbDecompXML 存储过程的 *documentid* 输入参数中指定的字符串值，此值标识要分解的 XML 文档。分解文档时，将把提供给 xdbDecompXML 存储过程的输入值用作 $DECOMP_DOCUMENTID 的替换值。

应用程序可以将生成的唯一文档标识传递给 xdbDecompXML。然后，可以将这些标识直接插入到数据库中。也可以将这些标识传递到生成唯一元素标识或属性标识的表达式中。因此，可使用 $DECOMP_DOCUMENTID 来插入 XML 文档未包含的唯一标识。

$DECOMP_ELEMENTID

系统生成的整数标识，它在 XML 文档中唯一地标识此注释所描述的元素或属性。只要未通过任何下列方法更改文档，此值在同一个 XML 文档的分解操作期间就保持不变：添加元素、删除元素或者更改元素在文档顺序中的位置。如果通过这些方法修改了文档，并且再次进行分解，元素的标识就可能与上次分解后的标识不同。对属性指定的 $DECOMP_ELEMENTID 被定义成此属性所属元素的 $DECOMP_ELEMENTID 值。

也可以使用 $DECOMP_ELEMENTID 生成的值来指示原始文档中的元素顺序。当需要根据关系表重新组成 XML 文档时，此关键字十分有用。

带注释的 XML 模式分解中如何形成分解结果

虽然典型的分解过程只分解 XML 元素或属性内容，但带注释的 XML 模式分解支持插入不必存在于 XML 文档中的值。

已分解的内容可以是下列任何一项:

- XML 文档中属性的值
- XML 文档中元素的值，其中准确的内容取决于 <db2-xdb:contentHandling> 注释的设置:
  - 文本 － 仅此元素中的字符数据（不包含其后代）
  - 字符串值 － 此元素中的字符数据及其后代
  - serializedSubtree － 此元素的开始标记（tag）和结束标记之间所有内容的标记（markup）

  有关更多信息，请参阅 <db2-xdb:contentHandling> 文档。
- 取决于 XML 文档中已映射的属性或元素的内容的值
- 独立于 XML 文档中的任何值的已生成值

通过 db2-xdb:expression 注释可以获得后面两种值。此注释允许您指定一个表达式，在分解期间将插入该表达式的结果。

可以将 XML 文档中的值应用于表达式以生成结果，从而在将数据插入到目标列之前变换数据。表达式还可以生成基于已映射的元素或属性值（例如，元素的名称）的值。db2-xdb:expression 还允许指定常量，其中常量可以与 XML 文档中已映射的值相关，也可以无关。db2-xdb:expression 允许您组合其中任意技巧来生成用于插入的值。

请注意，调用该表达式的次数与调用 XML 文档中遇到的与该表达式关联的元素或属性的次数相同。

对 XML 分解结果进行验证的作用

带注释的 XML 模式分解不要求验证输入文档，但建议您在分解之前或者分解期间执行验证，这样做有许多优点。

可以使用 XMLVALIDATE SQL/XML 函数在分解之前执行验证，也可以调用 xdbDecompXML 存储过程或 DECOMPOSE XML DOCUMENT 命令在分解期间执行验证。通过验证所分解的 XML 文档，可以确保：

- 仅当整个文档根据指定的 XML 模式有效时才会将值分解到表中（这确保仅将有效值存储在数据库中）
- 将为元素或属性定义的缺省值插入到数据库中（当使用其中一个 xdbDecompXML 分解存储过程执行验证并且 XML 文档未包含该元素或属性时）
- 将解析 XML 文档中的所有实体，以便在分解期间执行验证（如果 XML 文档中的某个实体在分解前尚未注册，就会返回错误）
- 根据模式中的指定执行非缺省空格规范化（当使用其中一个 xdbDecompXML 分解存储过程执行验证时）

由于分解过程假定输入文档对于相应带注释的模式来说是有效的，所以建议您针对已注册的 XML 模式验证输入文档。如果未执行验证，并且输入文档无效，分解过程就会对同一输入文档插入不同的行（这样做的目的是解析实体或添加缺省属性，这与执行验证时的情况不同），分解过程也可能会产生意外的结果。分解无效文档时产生的结果以及对现有数据的副作用是不确定的。

请注意，在分解期间执行验证时，模式错误（例如内容模型未确定）或者不正确的类型派生会导致分解过程失败。请验证带注释的模式是否正确，并在尝试再次进行分解前重新注册该模式。

带注释的 XML 模式分解中对 CDATA 部分的处理

对于为进行分解而添加了注释的元素来说，将把 CDATA 部分的内容插入到数据库中。还可以插入 CDATA 部分定界符（“<![CDATA[”和“]]>”）。XML 解析器将对 CDATA 内容进行行结束规范化处理。

在没有 db2-xdb:contentHandling 属性的情况下，对于为进行分解而添加了注释的元素来说，将把 CDATA 部分的内容插入到数据库中。未插入 CDATA 部分定界符。

如果使用 db2-xdb:contentHandling=″serializeSubtree″ 属性对 XML 模式中的 XML 元素声明添加了注释，那么在分解未解析的 XML 文档时将插入 CDATA 部分（包括 CDATA 定界符）。

未解析的 XML 与已解析的 XML 在分解结果上的差别

当使用 db2-xdb:contentHandling=″serializeSubtree″ 属性对相应的元素声明添加了注释时，分解 CDATA 部分将存在差别。分解结果取决于输入 XML 文档是未解析的 XML 还是已解析的 XML。例如，XML 文档作为未解析的 XML 存储在 CLOB 列，而作为已解析的 XML 存储在 XML 列中。

如果输入 XML 文档来自于非 XML 类型列，那么分解结果将保留元素中任何 CDATA 部分的边界和原始内容。如果输入文档来自于 XML 类型列，那么在分解结果中不会保留 CDATA 部分。以包含以下片段的 XML 文档为例:

```
<a> before cdata <![CDATA[ in cdata & <>]]> after cdata</a>
```

如果文档存储在 CLOB 列中，那么在元素 a 的映射中指定了 db2-xdb:contentHandling="serializeSubtree" 时，分解操作将对已映射至元素 a 的列生成以下结果:

```
<a>before cdata <![CDATA[ in cdata & <>]]> after cdata</a>
```

如果 XML 文档存储在 XML 类型列中，那么此 XML 片段的分解结果为:

```
<a> before cdata in cdata &amp; < &gt; after cdata</a>
```

只要用于分解的输入文档来自于 XML 类型列，就不会保留原来的 CDATA 部分。尽管不保留原来的 CDATA 部分与正确性问题无关（因为替代 CDATA 部分在逻辑上等价于原来的 CDATA 部分），但它可能会与期望的输出不同。

带注释的 XML 模式分解中的空值和空字符串

带注释的 XML 模式分解只有在某些情况下才会插入空值或空字符串。

XML 元素

下表说明了对于 XML 文档中的元素，在什么情况下会将空字符串或空值插入到数据库中。

表 70. 所映射元素的空值处理方式

条件	空字符串	空值
元素未包含在文档中		X
元素满足下列全部条件: • 包含在文档中 • 在 start 标记中包含 `xsi:nil="true"` 或 `xsi:nil="1"` 属性		X
元素满足下列全部条件: • 包含在文档中，并且是空的 • 在 start 标记中未包含 `xsi:nil="true"` 或 `xsi:nil="1"` 属性 • 派生自或者被声明为具有列表类型、联合类型、带有混合内容的复杂类型或者下列原子内置类型：xsd:string, xsd:normalizedString, xsd:token, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:anySimpleType；任何其他类型都将导致错误。	X	
注: 1. 如果某个映射涉及 db2-xdb:condition 或 db2-xdb:expression 注释，那么将传递空字符串或空值作为参数来进行表达式求值。 2. 如果目标列类型为 CHAR 或 GRAPHIC，那么将插入空字符串来作为空白字符的字符串。		

XML 属性

下表说明了当文档中添加了分解注释的 XML 属性包含空值或者该文档未包含该属性时，在什么情况下会将空字符串或空值插入到数据库中。

表 71. 所映射属性的空值处理方式

条件	空字符串	空值
属性未包含在文档中（由于未执行验证，或者由于验证操作未提供缺省值）		X
属性满足下列全部条件: • 包含在文档中，并且是空的 • 派生自或者被声明为具有列表类型、联合类型或者下列原子内置类型：xsd:string, xsd:normalizedString, xsd:token, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:anySimpleType;; 任何其他类型都将导致错误。	X	
注: 如果某个映射涉及 db2-xdb:condition 或 db2-xdb:expression 注释，那么将传递空字符串或空值作为参数来进行表达式求值。		

用于带注释的 XML 模式分解的核对表

带注释的 XML 模式分解会变得很复杂。为了使任务更好管理，应注意一些事项。

带注释的 XML 模式分解可能要求您将多个 XML 元素和属性映射至数据库中多个列和表。此映射还可能涉及在插入 XML 数据前对其进行变换或者应用插入条件。

以下是对 XML 模式添加注释时要考虑的事项以及相关文档的链接:

- 了解哪些分解注释可用。
- 在映射期间，确保列类型与所映射的元素或属性的 XML 模式类型兼容。
- 构造 XML 模式以最大程度地降低系统内存资源需求。
- 确保正确地对限制或扩展所派生的复杂类型添加注释。
- 确认未违反任何分解局限性与限制。
- 在向 XSR 注册模式时，确保注释中引用的表和列存在。

为进行带注释的 XML 模式分解而对派生的复杂类型添加的注释

在对通过限制或扩展派生的复杂类型添加注释以便进行分解时，需要应用其他的映射。

通过限制派生

通过限制派生的复杂类型要求在派生类型的定义中重复基本类型中的公共元素和属性。因此，还必须在派生类型中包括基本类型中的分解注释。

通过扩展派生

在通过扩展派生的复杂类型的定义中，仅指定未包括在基本类型中的元素和属性。如果派生类型的分解映射与基本类型的映射不同，那么必须对基本类型添加分解注释以明确地将基本类型的映射与派生类型的映射区分开。

以下示例说明如何将通过扩展派生的类型 outOfPrintBookType 映射至与其基本类型 bookType 不同的表。请注意，在 bookType 基本类型中指定了 db2-xdb:locationPath 注释，以明确地区分适用于基本类型的映射以及适用于派生类型的映射。在本示例中，由于派生类型 outOfPrintType 的 <lastPublished> 和 <publisher> 元素仅涉及在单个映射中，所以这些元素不需要 db2-xdb:locationPath 注释。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:table>
        <db2-xdb:name>BOOKS</db2-xdb:name>
        <db2-xdb:rowSet>inPrintRowSet</db2-xdb:rowSet>
      </db2-xdb:table>
      <db2-xdb:table>
        <db2-xdb:name>OUTOFPRINT</db2-xdb:name>
        <db2-xdb:rowSet>outOfPrintRowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="books">
    <xs:complexType>
      <xs:choice>
        <xs:element name="book" type="bookType"
                    minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="outOfPrintBook" type="outOfPrintBookType"
                    minOccurs="0" maxOccurs="unbounded"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="bookType">
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer"/>
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="title" type="xs:string"
                  db2-xdb:locationPath="/books/book/@title"
                  db2-xdb:rowSet="inPrintRowSet" db2-xdb:column="TITLE">
      <xs:annotation>
        <xs:appinfo>
         <db2-xdb:rowSetMapping db2-xdb:locationPath="/books/outOfPrintBook/@title">
           <db2-xdb:rowSet>outOfPrintRowSet</db2-xdb:rowSet>
           <db2-xdb:column>TITLE</db2-xdb:column>
         </db2-xdb:rowSetMapping>
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="isbn" type="xs:string"
                  db2-xdb:locationPath="/books/book/@isbn"
                  db2-xdb:rowSet="inPrintRowSet" db2-xdb:column="ISBN">
      <xs:annotation>
        <xs:appinfo>
         <db2-xdb:rowSetMapping db2-xdb:locationPath="/books/outOfPrintBook/@isbn">
           <db2-xdb:rowSet>outOfPrintRowSet</db2-xdb:rowSet>
           <db2-xdb:column>ISBN</db2-xdb:column>
         </db2-xdb:rowSetMapping>
        </xs:appinfo>
      </xs:annotation>
```

```
      </xs:attribute>
    </xs:complexType>
    <xs:complexType name="outOfPrintBookType">
      <xs:complexContent>
        <xs:extension base="bookType">
          <xs:sequence>
            <xs:element name="lastPublished" type="xs:date"
                     db2-xdb:rowSet="outOfPrintRowSet" db2-xdb:column="LASTPUBDATE"/>
            <xs:element name="publisher" type="xs:string"
                     db2-xdb:rowSet="outOfPrintRowSet" db2-xdb:column="PUBLISHER"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:simpleType name="paragraphType">
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:complexType name="chapterType">
      <xs:sequence>
        <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                 db2-xdb:locationPath="/books/book/chapter/paragraph"
                 db2-xdb:rowSet="inPrintRowSet" db2-xdb:column="CONTENT">
          <xs:annotation>
            <xs:appinfo>
              <db2-xdb:rowSetMapping
                db2-xdb:locationPath="/books/outOfPrintBook/chapter/paragraph">
                <db2-xdb:rowSet>outOfPrintBook</db2-xdb:rowSet>
                <db2-xdb:column>CONTENT</db2-xdb:column>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="number" type="xs:integer"/>
      <xs:attribute name="title" type="xs:string"/>
    </xs:complexType>
</xs:schema>
```

这些注释指示将把 <book> 元素中的值分解到 BOOKS 表中，并且将把 <outOfPrintBook> 元素中的值分解到 OUTOFPRINT 表中。

考虑 XML 文档中的以下元素:

```
<books>
  <book isbn="1-11-111111-1" title="My First XML Book">
    <authorID>22</authorID>
    <chapter number="1" title="Introduction to XML">
      <paragraph>XML is fun...</paragraph>
    </chapter>
    <chapter number="2" title="XML and Databases">
      <paragraph>XML can be used with...</paragraph>
    </chapter>
  </book>
  <outOfPrintBook isbn="7-77-777777-7" title="Early XML Book">
    <authorID>41</authorID>
    <chapter number="1" title="Introductory XML">
      <paragraph>Early XML...</paragraph>
    </chapter>
    <chapter number="2" title="What is XML">
      <paragraph>XML is an emerging technology...</paragraph>
    </chapter>
    <lastPublished>2000-01-31</lastPublished>
    <publisher>Early Publishers Group</publisher>
  </outOfPrintBook>
</books>
```

使用上述带注释的模式分解此元素所属的文档将产生下列各表:

表 72. BOOKS

ISBN	TITLE	CONTENT
1-11-111111-1	My First XML Book	XML is fun...
1-11-111111-1	My First XML Book	XML can be used with...

表 73. OUTOFPRINT

ISBN	TITLE	CONTENT	LASTPUBDATE	PUBLISHER
7-77-777777-7	Early XML Book	Early XML...	2000-01-31	Early Publishers Group
7-77-777777-7	Early XML Book	XML is an emerging technology...	2000-01-31	Early Publishers Group

分解功能的 XML 模式构造建议

通过调整带注释的 XML 模式中的元素顺序，可以将带注释的模式分解对系统内存资源的需求降至最低。

对于非常大的文档来说，遵循此建议可能使得不必增加 DB2 数据库服务器的可用内存量，即可分解文档。对于为进行分解而添加了注释的同代元素来说，应该将简单类型的元素放在带注释的模式中同代复杂类型元素前面。同样，应该将 maxOccurs 属性设置为 1 的同代元素放在 maxOccurs > 1 的同代元素前面。

在处理完构成一行的所有项之前，必须将每个构成该行的项都存放在内存中，因此，带注释的模式分解需要耗用的内存量受 XML 模式结构影响。这些模式构造建议对一行的各个项进行组织，以将必须存放在内存中的项数降至最低。

以下示例说明了所映射同代元素的建议 XML 模式构造与非最优构造之间的对比。请注意，在非最优示例中，复杂类型的 <complexElem> 放在简单类型的 <status> 前面。通过将 <complexElem> 放在 <id> 和 <status> 元素后面，可以提高分解运行时效率。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
  <-- Recommended structuring with simple types placed before
      the recurring element <wrapper>, which is of complex type -->
  <xs:complexType name="typeA">
    <xs:sequence>
      <xs:element name="id" type="xs:integer"
                  db2-xdb:rowSet="relA" db2-xdb:column="ID" />
      <xs:element name="status" type="xs:string"
                  db2-xdb:rowSet="relA" db2-xdb:column="status" />
      <xs:element name="wrapper" type="typeX" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <-- Less optimal structuring with recurring complex type element
      appearing before the simple type element -->
  <--
  <xs:complexType name="typeA">
    <xs:sequence>
      <xs:element name="id" type="xs:integer"
                  db2-xdb:rowSet="relA" db2-xdb:column="ID" />
      <xs:element name="wrapper" type="typeX" maxOccurs="unbounded"/>
      <xs:element name="status" type="xs:string"
                  db2-xdb:rowSet="relA" db2-xdb:column="status" />
```

```
    </xs:sequence>
  </xs:complexType> -->

  <xs:complexType name="typeX">
    <xs:sequence>
      <xs:element name="elem1" type="xs:string"
                  db2-xdb:rowSet="relA" db2-xdb:column="elem1" />
      <xs:element name="elem2" type="xs:long"
                  db2-xdb:rowSet="relA" db2-xdb:column="elem2" />
    </xs:sequence>
  </xs:complexType>

  <xs:element name="A" type="typeA" />

</xs:schema>
```

请注意，<id>、<status>、<elem1> 和 <elem2> 映射至同一个行集，也就是说，它们共同构成一行。一行完成后，就会释放与该行相关联的内存。在上面给出的非最优情况下，在到达文档中的 <status> 元素之前，不能将任何与行集 relA 相关联的行视为已完成。但是，由于 <wrapper> 元素在 <status> 元素之前出现，所以必须首先处理 <wrapper> 元素。这意味着，在到达 <status> 元素之前（或者，如果文档未包含 <status>，那么在到达 <A> 末尾之前），必须将 <wrapper> 的所有实例都缓存在内存中。

如果某个元素有大量的实例，那么这种结构的影响重大。例如，如果 <wrapper> 元素有 10 000 个实例，那么在行集完成前，必须将这 10 000 个实例都存放在内存中。但是，在上面给出的最优情况下到达 <elem2> 时，可以释放与行集 relA 的行相关联的内存。

带注释的 XML 模式分解中的映射示例

带注释的 XML 模式分解依靠映射确定将 XML 文档分解为表的方式。映射表示成注释的形式添加到 XML 模式文档中。这些映射描述要将 XML 文档分解为表的方式。下列示例显示一些常见的映射方案。

常见映射方案:

带注释的 XML 模式分解中的行集

db2-xdb:rowSet 标识将值分解到其中的目标表。可以将此注释设置为表名或行集名。

行集是使用 db2-xdb:rowSet 注释指定的，它将作为元素或属性声明的属性或 <db2-xdb:rowSetMapping> 注释的子注释添加到 XML 模式文档。

组成 XML 模式的所有模式文档中对元素或属性的实例具有相同 db2-xdb:rowSet 值的一组映射定义一行。

例如，考虑以下 XML 文档:

```
<publications>
  <textbook title="Programming with XML">
        <isbn>0-11-011111-0</isbn>
        <author>Mary Brown</author>
        <author>Alex Page</author>
        <publicationDate>2002</publicationDate>
        <university>University of London</university>
    </textbook>
    <childrensbook title="Children's Fables">
        <isbn>5-55-555555-5</isbn>
        <author>Bob Carter</author>
```

```
        <author>Melaine Snowe</author>
        <publicationDate>1999</publicationDate>
    </childrensbook>
</publications>
```

要分解此文档，以便每本书的 ISBN 和标题（无论是课本还是儿童书）都插入到同一个表（名称为 ALLPUBLICATIONS）中，必须定义多个行集：一个行集用于分组与课本相关的值，而另一个行集用于分组与儿童书相关的值。

在这种情况下，行集确保只有那些语义相关的值才分组在一起形成一行。也就是说，使用行集将通过课本的标题对课本的 ISBN 值进行分组，并通过儿童书的标题对儿童书的 ISBN 进行分组。这确保一行中将不包含课本的 ISBN 值，但具有儿童书的标题。

如果没有行集，那么无法确定哪些值应该分组在一起以形成语义仍正确的行。

接下来说明行集在 XML 模式文档中的应用。分别在 <textbook> 和 <childrensbook> 元素的 ISBN 元素声明中指定了两个行集 textbk_rowSet 和 childrens_rowSet。然后，将这些行集通过 <db2-xdb:table> 注释与 ALLPUBLICATIONS 表关联。

请注意，不要将行集注释用作表标识，而是用作行集标识，这样您可以很容易更改在 XML 模式中引用的表名。这是因为当 db2-xdb:rowSet 的值表示标识而不是表名时，需要使用 <db2-xdb:table><db2-xdb:name></db2-xdb:name></db2-xdb:table> 注释来真正指定表名。借助此方法，您只需要在必要时更新一个位置的表名。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>admin</db2-xdb:defaultSQLSchema>
      <db2-xdb:table>
        <db2-xdb:name>ALLPUBLICATIONS</db2-xdb:name>
        <db2-xdb:rowSet>textbk_rowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>childrens_rowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>
    <xs:element name="publications">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="textbook" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="isbn" type="xs:string"
                       db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_ISBN"/>
                <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="publicationDate" type="xs:gYear"/>
                <xs:element name="university" type="xs:string"
                            maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute name="title" type="xs:string" use="required"
                     db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_TITLE"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="childrensbook" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="isbn" type="xs:string"
                    db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_ISBN"/>
                <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="publicationDate" type="xs:gYear"/>
              </xs:sequence>
```

```
            <xs:attribute name="title" type="xs:string" use="required"
                 db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_TITLE"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

下面显示了通过使用此带注释的 XML 模式进行分解产生的表。

表 74. ALLPUBLICATIONS

ISBN	PUBS_TITLE
0-11-011111-0	Programming with XML
5-55-555555-5	Children's Fables

虽然上面的示例说明了使用行集进行分解的简单情况，但可以在更复杂的映射中使用行集来将 XML 模式的不同部分中的多个项分组在一起，以形成表和列对相同的行。

条件变换

行集允许您根据正被分解的值本身对这些值应用不同的变换。

例如，考虑名为“temperature”的元素的下面两个实例:

```
<temperature unit="Celsius">49</temperature>
<temperature unit="Fahrenheit">49</temperature>
```

如果这些元素的值将插入到同一个表中，并且您想要该表包含一致的值（例如，全部为 Celsius 值），那么在插入之前，需要将具有属性 unit="Fahrenheit" 的值转换为 Celsius。可以通过将属性为 unit="Celsius" 的所有元素映射至一个行集，并将属性为 unit="Fahrenheit" 的所有元素映射至另一个行集来实现此操作。然后，可以在插入之前对 Fahrenheit 值的行集应用转换公式。

注意，“unit”的属性声明的映射不包含任何 db2-xdb:column 规范。这意味着该项的值仅用于条件求值，而不会用于存储到 db2-xdb:rowSet 规范指定的表中。

可使用以下 XML 模式文档将 Celsius 和已转换的 Fahrenheit 值插入到同一个表中:

```
....
<!-- Global annotation -->
<db2-xdb:table>
  <db2-xdb:name>TEMPERATURE_DATA</db2-xdb:name>
  <db2-xdb:rowSet>temp_celsius</db2-xdb:rowSet>
  <db2-xdb:rowSet>temp_fahrenheit</db2-xdb:rowSet>
</db2-xdb:table>
...
<xs:element name="temperature">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>temp_celsius</db2-xdb:rowSet>
        <db2-xdb:column>col1</db2-xdb:column>
      </db2-xdb:rowSetMapping>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>temp_fahrenheit</db2-xdb:rowSet>
        <db2-xdb:column>col1</db2-xdb:column>
        <db2-xdb:expression>
   myudf_convertToCelsius($DECOMP_CONTENT)
```

```
</db2-xdb:expression>
     </db2-xdb:rowSetMapping>
   </xs:appinfo>
 </xs:annotation>
 <xs:complexType>
   <xs:simpleContent>
     <xs:extension base="xs:int">
       <xs:attribute name="unit" type="xs:string">
         <xs:annotation>
           <xs:appinfo>
             <db2-xdb:rowSetMapping>
               <db2-xdb:rowSet>temp_celsius</db2-xdb:rowSet>
               <db2-xdb:condition>
   $DECOMP_CONTENT = 'Celsius'
 </db2-xdb:condition>
             </db2-xdb:rowSetMapping>
             <db2-xdb:rowSetMapping>
               <db2-xdb:rowSet>temp_fahrenheit</db2-xdb:rowSet>
               <db2-xdb:condition>
   $DECOMP_CONTENT = 'fahrenheit'
 </db2-xdb:condition>
             </db2-xdb:rowSetMapping>
           </xs:appinfo>
         </xs:annotation>
       </xs:attribute>
     </xs:extension>
   </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

分解注释示例: 映射至 XML 列

在带注释的 XML 模式分解中，可以将 XML 片段映射至使用 XML 数据模型定义的列。

考虑以下 XML 文档:

```
<publications>
  <textbook title="Programming with XML">
    <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
</publications>
```

如果想要按如下所示存储 XML 元素 <textbook> 和书名，那么需要在相应 XML 模式文档中对 <textbook> 元素和 title 属性的声明添加注释。这些注释应指定 DETAILS 和 TITLE 列（其中 DETAILS 列已定义为 XML 类型）以及 TEXTBOOKS 表。

表 75. TEXTBOOKS

TITLE	DETAILS

Programming with XML	`<textbook title="Programming with XML">`
	`  <isbn>0-11-011111-0</isbn>`
	`  <author>Mary Brown</author>`
	`  <author>Alex Page</author>`
	`  <publicationDate>2002</publicationDate>`
	`  <university>University of London</university>`
	`</textbook>`

可以根据注释在模式文档中将注释指定为属性或元素。某些注释可以指定为任意一种。请参阅每个具体注释的文档以确定可以如何指定特定注释。

通过将 db2-xdb:rowSet 和 db2-xdb:column 用作 <xs:element> 或 <xs:attribute> 的属性或者使用 <db2-xdb:rowSetMapping> 的 <db2-xdb:rowSet> 和 <db2-xdb:column> 子元素来指定目标表和列。将这些映射指定为元素或属性都一样。

以下 XML 模式文档片段说明了如何通过将注释指定为属性来将两个映射添加至 <textbook> 元素和 title 属性。

```
<xs:element name="publications">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="textbook" maxOccurs="unbounded"
                 db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="DETAILS">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="isbn" type="xs:string"/>
            <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="publicationDate" type="xs:gYear"/>
            <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
          </xs:sequence>
         <xs:attribute name="title" type="xs:string" use="required"
                       db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="TITLE"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

db2-xdb:rowSet 注释指定目标表的名称，而 db2-xdb:column 注释指定目标列的名称。因为 <textbook> 元素是复杂类型并且包含复杂内容，而且没有指定 db2-xdb:contentHandling 注释，所以在缺省情况下，将按照 db2-xdb:contentHandling 的 serializeSubtree 设置把该元素中的所有标记（包括它的开始标记和结束标记）插入到 XML 列中。将保留 XML 文档中的空格。有关更多详细信息，请参阅 db2-xdb:contentHandling 文档。

分解注释示例：一个值映射至单个表会产生单个行

将 XML 文档中的一个值映射至单个表和列对是带注释的 XML 模式分解中一种简单形式的映射。此示例显示行集中的值之间较简单的一对一关系的情况。

此映射的结果取决于映射至同一行集的项之间的关系。如果一起映射至单个行集中的值具有一对一关系，正如元素的 maxOccurs 属性的值或包含的模型组声明所确定的那样，将对 XML 文档中映射的项的每个实例形成单个行。如果单个行集中的值具有一对多关系（即，一个值对于另一个项的多个实例在文档中仅出现一次），正如 maxOccurs 属性的值所指示的那样，那么分解 XML 文档时将产生多个行。

考虑以下 XML 文档:

```
<publications>
  <textbook title="Programming with XML">
    <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
</publications>
```

如果想要 <isbn> 和 <publicationDate> 元素的值以及 title 属性分解为如下所示的 TEXTBOOKS 表，那么需要在相应 XML 模式文档中这些元素和属性的声明中添加注释。注释将指定每个项映射至的表名和列名。

表 76. TEXTBOOKS

ISBN	TITLE	DATE
0-11-011111-0	Programming with XML	2002

可以根据注释在模式文档中将注释指定为属性或元素。某些注释可以指定为任意一种。请参阅每个具体注释的文档以确定可以如何指定特定注释。

对于将一个值映射至单个表和列对的情况，需要对映射的值指定表和列。这是使用 db2-xdb:rowSet 和 db2-xdb:column 作为 <xs:element> 或 <xs:attribute> 的属性或使用 <db2-xdb:rowSetMapping> 的 <db2-xdb:rowSet> 和 <db2-xdb:column> 子元素完成的。将这些映射指定为元素或属性都一样。

以下示例说明如何通过将注释指定为属性来把 <textbook> 元素中的元素和属性映射至 TEXTBOOKS 表。

```
<xs:element name="publications">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="textbook" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="isbn" type="xs:string"
                        db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="ISBN"/>
            <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="publicationDate" type="xs:gYear"
                        db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="DATE"/>
            <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
          </xs:sequence>
         <xs:attribute name="title" type="xs:string" use="required"
                       db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="TITLE"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  </xs:element>
```

XML 模式属性 maxOccurs 的缺省值为 1，因此，映射至 TEXTBOOKS 行集的每个项彼此之间具有一对一关系。因为这种一对一关系，所以会为 <textbook> 元素的每个实例形成单个行。

分解注释示例：一个值映射至单个表会产生多个行

将 XML 文档中的一个值映射至单个表和列对是带注释的 XML 模式分解中一种简单形式的映射。此示例显示行集中的值之间较复杂的一对多关系的情况。

此映射的结果取决于映射至同一行集的项之间的关系。如果一起映射至单个行集中的值具有一对一关系，正如元素的 maxOccurs 属性的值或包含的模型组声明所确定的那样，将对 XML 文档中映射的项的每个实例形成单个行。如果单个行集中的值具有一对多关系（即，一个值对于另一个项的多个实例在文档中仅出现一次），正如 maxOccurs 属性的值所指示的那样，那么分解 XML 文档时将产生多个行。

考虑以下 XML 文档：

```
<textbook title="Programming with XML">
      <isbn>0-11-011111-0</isbn>
      <author>Mary Brown</author>
      <author>Alex Page</author>
      <publicationDate>2002</publicationDate>
      <university>University of London</university>
</textbook>
```

如果想要按如下所示存储课本的 ISBN 和作者，那么要在相应的 XML 模式文档中对 <isbn> 和 <author> 元素的声明添加注释。注释应指定 ISBN 和 AUTHNAME 列以及 TEXTBOOK_AUTH 表。

表 77. TEXTBOOKS_AUTH

ISBN	AUTHNAME
0-11-011111-0	Mary Brown
0-11-011111-0	Alex Page

可以根据注释在模式文档中将注释指定为属性或元素。某些注释可以指定为任意一种。请参阅每个具体注释的文档以确定可以如何指定特定注释。

对于将一个值映射至单个表和列对的情况，需要对映射的值指定表和列。这是使用 db2-xdb:rowSet 和 db2-xdb:column 作为 <xs:element> 或 <xs:attribute> 的属性或使用 <db2-xdb:rowSetMapping> 的 <db2-xdb:rowSet> 和 <db2-xdb:column> 子元素完成的。

将这些映射指定为元素或属性都一样。在下面出现的 XML 模式文档中将映射指定为元素。

```
<xs:element name="textbook" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isbn" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>TEXTBOOKS_AUTH</db2-xdb:rowSet>
              <db2-xdb:column>ISBN</db2-xdb:column>
            </db2-xdb:rowSetMapping>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="author" type="xs:string" maxOccurs="unbounded">
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>TEXTBOOKS_AUTH</db2-xdb:rowSet>
              <db2-xdb:column>AUTHNAME</db2-xdb:column>
            </db2-xdb:rowSetMapping>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="publicationDate" type="xs:gYear"/>
      <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="title" type="xs:string"
use="required"/>
  </xs:complexType>
  </xs:element>
```

请注意，为什么 <isbn> 元素只映射至 ISBN 列一次，却出现在表的两行中呢？这是在分解过程中自动发生的，因为每个 ISBN 值有多个作者。<isbn> 的值在每个作者对应的每一行中重复。

出现这种情况是因为在 <isbn> 和 <author> 元素之间检测到一对多关系，因为 <author> 的 maxOccurs 属性大于 1。

请注意，一对多关系可以涉及两个以上的项，并包括多组项。一对多关系还可以深深嵌套，其中在一个一对多关系中已涉及的项可以参与另一个一对多关系。

分解注释示例: 一个值映射至多个表

可以将 XML 文档中的单个值映射至多个表。此示例说明如何注释 XML 模式文档以将一个值映射至两个表。

考虑以下 XML 文档。

```
<textbook title="Programming with XML">
      <isbn>0-11-011111-0</isbn>
      <author>Mary Brown</author>
      <author>Alex Page</author>
      <publicationDate>2002</publicationDate>
      <university>University of London</university>
</textbook>
```

要将课本的 ISBN 映射至下列两个表，需要对 <isbn> 元素创建两个映射。这可以通过在 XML 模式文档中将多个 <db2-xdb:rowSetMapping> 元素添加至 <isbn> 元素声明来实现。

表 78. TEXTBOOKS

ISBN	TITLE
0-11-011111-0	Programming with XML

表 79. SCHOOLPUBS

ISBN	SCHOOL
0-11-011111-0	University of London

以下 XML 模式文档片段说明如何将两个映射添加至 <isbn> 元素声明来指定至两个表的映射。title 属性和 <university> 元素的值也包括在这些映射中。

```
<xs:element name="textbook" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isbn" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>TEXTBOOKS</db2-xdb:rowSet>
              <db2-xdb:column>ISBN</db2-xdb:column>
            </db2-xdb:rowSetMapping>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>SCHOOLPUBS</db2-xdb:rowSet>
              <db2-xdb:column>ISBN</db2-xdb:column>
            </db2-xdb:rowSetMapping>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
```

```
        <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
        <xs:element name="publicationDate" type="xs:gYear"/>
        <xs:element name="university" type="xs:string" maxOccurs="unbounded">
          <xs:annotation>
            <xs:appinfo>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>SCHOOLPUBS</db2-xdb:rowSet>
                <db2-xdb:column>SCHOOL</db2-xdb:column>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="title" type="xs:string" use="required">
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>TEXTBOOKS</db2-xdb:rowSet>
              <db2-xdb:column>TITLE</db2-xdb:column>
            </db2-xdb:rowSetMapping>
          </xs:appinfo>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
```

多次出现的复杂类型

如果 XML 模式中的多个位置引用一种复杂类型，那么可以根据该类型在模式中的位置使用 db2-xdb:locationPath 注释将它映射至不同的表和列。

在这种情况下，需要使用多个 <db2-xdb:rowSetMapping> 注释（每个映射一个注释）来为该复杂类型元素或属性声明添加注释，其中每个映射由 db2-xdb:locationPath 属性区分。

分解注释示例：将映射至单个表的多个值进行分组

在带注释的 XML 模式分解中，可以将不相关的元素中的多个值映射至同一个表，同时保持逻辑相关的值之间的关系。通过声明多个用于分组相关项以形成一行的行集，可以实现该目的，如以下示例中所示。

例如，考虑以下 XML 文档：

```
<publications>
  <textbook title="Programming with XML">
        <isbn>0-11-011111-0</isbn>
        <author>Mary Brown</author>
        <author>Alex Page</author>
        <publicationDate>2002</publicationDate>
        <university>University of London</university>
    </textbook>
    <childrensbook title="Children's Fables">
        <isbn>5-55-555555-5</isbn>
        <author>Bob Carter</author>
        <author>Melaine Snowe</author>
        <publicationDate>1999</publicationDate>
    </childrensbook>
</publications>
```

要在分解后生成下表，需要确保与课本相关的值和与儿童书关联的值不分组到同一行中。使用多个行集来分组相关的值并生成在逻辑上有意义的行。

表 80. ALLPUBLICATIONS

PUBS_ISBN	PUBS_TITLE
0-11-011111-0	Programming with XML
5-55-555555-5	Children's Fables

在将单个值映射至单个表和列对的简单映射方案中，可以只指定要将值映射至的表和列。

然而，此示例显示的情况更加复杂，它将多个值映射至同一个表，且必须对这些值进行逻辑分组。如果您只是要将每个 ISBN 和标题映射至 PUBS_ISBN 和 PUBS_TITLE 列，而且不使用行集，那么分解过程将无法确定哪个 ISBN 值属于哪个标题值。通过使用行集，可以将逻辑相关的值进行分组以形成一个有意义的行。

以下 XML 模式文档说明如何定义两个行集来区分 <textbook> 元素的值与 <childrensbook> 元素的值。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:table>
        <db2-xdb:name>ALLPUBLICATIONS</db2-xdb:name>
        <db2-xdb:rowSet>textbk_rowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>childrens_rowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="publications">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="textbook" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="isbn" type="xs:string"
                         db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_ISBN"/>
              <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
              <xs:element name="publicationDate" type="xs:gYear"/>
              <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="title" type="xs:string" use="required"
                         db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_TITLE"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="childrensbook" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="isbn" type="xs:string"
                     db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_ISBN"/>
              <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
              <xs:element name="publicationDate" type="xs:gYear"/>
            </xs:sequence>
            <xs:attribute name="title" type="xs:string" use="required"
                     db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_TITLE"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

请注意，db2-xdb:rowSet 在每个元素和属性声明中的映射方式不指定表名，而是指定行集名。将行集与 <db2-xdb:table> 注释中的 ALLPUBLICATIONS 表关联，该注释必须指定为 <xs:schema> 的子元素。

通过指定映射至同一个表的多个行集，可以确保逻辑相关的值在表中形成一行。

分解注释示例：将不同上下文中的多个值映射至单个表

在带注释的 XML 模式分解中，可以将多个值映射至同一个表和列，以便单个列中可以包含来自文档不同部分的值。这可以通过声明多个行集来实现，如以下示例中所示。

例如，考虑以下 XML 文档：

```
<publications>
  <textbook title="Principles of Mathematics">
    <isbn>1-11-111111-1</isbn>
    <author>Alice Braun</author>
    <publisher>Math Pubs</publisher>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
</publications>
```

可以将作者和出版社映射至同一个包含特定书的联系人的表。

表 81. BOOKCONTACTS

ISBN	CONTACT
1-11-111111-1	Alice Braun
1-11-111111-1	Math Pubs

生成的表中 CONTACT 列的值来自 XML 文档的不同部分：一行可能包含作者的姓名（来自 <author> 元素），而另一个行包含出版社的名称（来自 <publisher> 元素）。

以下 XML 模式文档说明如何使用多个行集来生成此表。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:table>
        <db2-xdb:name>BOOKCONTACTS</db2-xdb:name>
        <db2-xdb:rowSet>author_rowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>publisher_rowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="publications">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="textbook" maxOccurs="unbounded">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="isbn" type="xs:string">
                 <xs:annotation>
                   <xs:appinfo>
                     <db2-xdb:rowSetMapping>
                       <db2-xdb:rowSet>author_rowSet</db2-xdb:rowSet>
                       <db2-xdb:column>ISBN</db2-xdb:column>
                     </db2-xdb:rowSetMapping>
                     <db2-xdb:rowSetMapping>
```

```
                <db2-xdb:rowSet>publisher_rowSet</db2-xdb:rowSet>
                <db2-xdb:column>ISBN</db2-xdb:column>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
        <xs:element name="author" type="xs:string" maxOccurs="unbounded">
          <xs:annotation>
            <xs:appinfo>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>author_rowSet</db2-xdb:rowSet>
                <db2-xdb:column>CONTACT</db2-xdb:column>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
        <xs:element name="publisher" type="xs:string">
          <xs:annotation>
            <xs:appinfo>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>publisher_rowSet</db2-xdb:rowSet>
                <db2-xdb:column>CONTACT</db2-xdb:column>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
        <xs:element name="publicationDate" type="xs:gYear"/>
        <xs:element name="university" type="xs:string"
                    maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="title" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

请注意，db2-xdb:rowSet 在每个元素声明中的映射方式不指定表名，而是指定行集的名称。将行集与 <db2-xdb:table> 注释中的 BOOKCONTACTS 表关联，该注释必须指定为 <xs:schema> 的子元素。

带注释的模式分解的 XML 模式到 SQL 类型兼容性

带注释的 XML 模式分解使您能够将 XML 值存储在表列中。XML 值只能分解为兼容 SQL 列。下表列示哪些 XML 模式类型与哪些 SQL 列类型相兼容。

表 82. 兼容的 XML 模式和 SQL 数据类型

XML 模式类型	SQL 类型																						
	SMALLINT	INTEGER	BIGINT	REAL	DECIMAL	DOUBLE	DECFLOAT(16)	DECFLOAT(34)	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR FBD†	BLOB	VARCHAR FBD†	LONG VARCHAR FBD†
string、normalizedString 和 token	1	1	1	1	1	1	1	1	2	3	4	6	5	5	5	6a	5a	5a	5a	7a	7	7	7
base64Binary 和 hexBinary	-	-	-	-	-	-	-	-	-	-	-	8a	8	8	8	-	-	-	-	8c	8b	8b	8b
byte 和 unsigned byte	0a	0a	0a	0a	0a	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
integer、positiveInteger、negativeInteger、nonNegativeInteger 和 nonPositiveInteger	10	10	10	11	11	11	10	10	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
int	10	0a	0a	11	11	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
unsignedInt	10	10	0a	11	11	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
long	10	10	0a	11	11	11	10	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
unsignedLong	10	10	10	11	11	11	10	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
short	0a	0a	0a	0a	0a	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
unsignedShort	10	0a	0a	0a	0a	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
decimal	21	21	21	11	11	11	11	11	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
float	22	22	22	17	16	17	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
double	22	22	22	16	16	17	11	11	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
boolean	12	12	12	12	12	12	12	12	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
time	-	-	-	-	-	-	-	-	-	14	-	13a*	13*	13*	13*	-	-	-	-	-	-	-	-
dateTime	-	-	-	-	-	-	-	-	15	15	19	13a*	13*	13*	13*	-	-	-	-	-	-	-	-
duration、gMonth、gYear、gDay、gMonthDay、gYearMonth	-	-	-	-	-	-	-	-	-	-	-	13a	13	13	13	-	-	-	-	-	-	-	-
date	-	-	-	-	-	-	-	-	20	-	-	13a*	13*	13*	13*	-	-	-	-	-	-	-	-
Name、NCName、NOTATION、ID、IDREF、QName、NMTOKEN、ENTITY	-	-	-	-	-	-	-	-	-	-	-	6	5	5	5	6a	5a	5a	5a	7a	7	7	7
ENTITIES、NMTOKENS、IDREFS 和列表类型	-	-	-	-	-	-	-	-	-	-	-	6b	5b	5b	5b	6c	5c	5c	5c	7c	7b	7b	7b
anyURI	-	-	-	-	-	-	-	-	-	-	-	18a	18	18	18	-	-	-	-	7a	7	7	7
language	-	-	-	-	-	-	-	-	-	-	-	6	5	5	5	-	-	-	-	7a	7	7	7
anySimpleType 和联合类型	-	-	-	-	-	-	-	-	-	-	-	6d	5d	5d	5d	6e	5e	5e	5e	7e	7d	7d	7d
anyType	-	-	-	-	-	-	-	-	-	-	-	6d	5d	5d	5d	6e	5e	5e	5e	7e	7d	7d	7d

图注

† FOR BIT DATA

***** db2-xdb:normalization 注释用于确定插入到数据库中的字符串的格式。

– 数据类型与带注释的 XML 模式分解不兼容。

0 数据类型是兼容的。

0a 兼容。并且，当 XML 类型的值空间包含 -0 时，-0 在数据库中存储为 0。

1 如果字符串采用的是目标 SQL 类型可接受的词汇格式，并且可以转换为在 SQL 类型的范围内的数值，那么兼容。但是可能会丢失有效数字。

2 如果字符串具有有效日期格式（*yyyy-mm-dd*、*mm/dd/yyyy* 或 *dd.mm.yyyy*），那么兼容。

3 如果字符串具有有效时间格式（*hh.mm.ss*、*hh:mm AM or PM* 或 *hh:mm:ss*），那么兼容。

4 如果字符串具有有效时间戳记格式（*yyyy-mm-dd-hh.mm.ss.nnnnnn* 或 *yyyy-mm-dd hh.mm.ss.nnnnnn*），那么兼容。

5 如果 XML 输入字符串长度（以字节计）小于或等于目标列长度（以字节计），那么兼容。如果输入字符串比目标列长，那么仅当对此列映射将 db2-xdb:truncate 设置为“true”或“1”时，字符串才兼容。字符串长度是在规范化后计算的，其中，根据 XML 模式类型的空格面来对输入字符串进行规范。

5a 根据 5 中描述的条件确定是否兼容。此外，输入字符串必须由双字节字符组成。

5b 根据 5 中描述的条件确定是否兼容。此外，插入到目标列中的值是并置的列表项字符串，各个列表项之间由单个空格分隔（与列表的“折叠”空格面一致）。

5c 根据 5a 中描述的条件确定是否兼容。此外，插入到目标列中的值是并置的列表项字符串，各个列表项之间由单个空格分隔（与列表的“折叠”空格面一致）。

5d 如果 XML 输入字符串长度（以字节计）小于或等于目标列长度（以字节计），那么兼容。如果输入字符串比目标列长，那么仅当对此列映射将 db2-xdb:truncate 设置为“true”或“1”时，字符串才兼容。无论哪一种情况，插入到目标列中的值都是元素或属性的字符内容。

5e 根据 5d 中描述的条件确定是否兼容。此外，输入字符串必须由双字节字符组成。

6 如果 XML 输入字符串长度（以字节计）小于或等于目标列长度（以字节计），那么兼容。如果输入字符串比目标列长，那么仅当对此列映射将 db2-xdb:truncate 设置为“true”或“1”时，字符串才兼容。字符串长度是在规范化后计算的，其中，根据 XML 模式类型的空格面来对输入字符串进行规范。如果输入 XML 字符串的长度小于目标列的定义长度，那么插入该字符串时将在右边填充空白。

6a 根据 6 中描述的条件确定是否兼容。此外，输入字符串必须由双字节字符组成。

6b 根据 6 中描述的条件确定是否兼容。此外，插入到目标列中的值是并置的列表项字符串，各个列表项之间由单个空格分隔（与列表的“折叠”空格面一致）。

6c 根据 6a 中描述的条件确定是否兼容。此外，插入到目标列中的值是并置的列表项字符串，各个列表项之间由单个空格分隔（与列表的“折叠”空格面一致）。

6d 如果 XML 输入字符串长度（以字节计）小于或等于目标列长度（以字节计），那么兼容。如果输入字符串比目标列长，那么仅当对此列映射将 db2-xdb:truncate 设置为“true”或“1”时，字符串才兼容。无论哪一种情况，插入到目标列中的值都是元素或属性的字符内容。如果输入 XML 字符串的长度小于目标列的定义长度，那么插入该字符串时将在右边填充空白。

6e 根据 6d 中描述的条件确定是否兼容。此外，输入字符串必须由双字节字符组成。

7 如果 XML 输入字符串长度（以字节计）小于或等于目标列长度（以字节计），那么兼容。如果输入字符串比目标列长，那么仅当对此列映射将 db2-xdb:truncate 设置为“true”或“1”时，字符串才兼容。字符串长度是在规范化后计算的，其中，根据 XML 模式类型的空格面来对输入字符串进行规范。

7a 根据 7 中描述的条件确定是否兼容。此外，如果输入 XML 字符串的长度小于目标列的定义长度，那么插入该字符串时将在右边填充空白。

7b 根据 7 中描述的条件确定是否兼容。此外，插入到目标列中的值是并置的列表项字符串，各个列表项之间由单个空格分隔（与列表的“折叠”空格面一致）。

7c 根据 7b 中描述的条件确定是否兼容。此外，如果输入 XML 字符串的长度小于目标列的定义长度，那么插入该字符串时将在右边填充空白。

7d 如果 XML 输入字符串长度（以字节计）小于或等于目标列长度（以字节计），那么兼容。如果输入字符串比目标列长，那么仅当对此列映射将 db2-xdb:truncate 设置为“true”或“1”时，字符串才兼容。无论哪一种情况，插入到目标列中的值都是元素或属性的字符内容。

7e 根据 7d 中描述的条件确定是否兼容。此外，如果输入 XML 字符串的长度小于目标列的定义长度，那么插入该字符串时将在右边填充空白。

8 如果 XML 输入字符串长度（以字节计）小于或等于目标列长度（以字节计），那么兼容。如果输入字符串比目标列长，那么仅当对此列映射将 db2-xdb:truncate 设置为“true”或“1”时，字符串才兼容。将插入已编码的（原始）字符串。

8a 根据 8 中描述的条件确定是否兼容。此外，如果输入 XML 字符串的长度小于目标列的定义长度，那么插入该字符串时将在右边填充空白。

8b 如果 XML 输入字符串长度（以字节计）小于或等于目标列长度（以字节计），那么兼容。如果输入字符串比目标列长，那么仅当对此列映射将 db2-xdb:truncate 设置为“true”或“1”时，字符串才兼容。插入到目标列中的值是已解码的字符串。

8c 根据 8b 中描述的条件确定是否兼容。此外，如果输入 XML 字符串的长度小于目标列的定义长度，那么插入该字符串时将在右边填充空白。

9 如果处理完成后根据 db2-xdb:normalization 设置计算的 XML 输入字符串长度小于或等于目标列长度，那么兼容。而且，如果对此列映射将 db2-xdb:truncate 设置为“true”或“1”，也兼容。

9a 根据 9 中描述的条件确定是否兼容。此外，如果输入 XML 字符串的长度小于目标列的定义长度，那么插入该字符串时将在右边填充空白。

10 如果 XML 类型在 SQL 类型的范围内，那么兼容。当 XML 类型的值空间包含 -0 时，-0 在数据库中存储为 0。

11 如果 XML 值在 SQL 类型的范围内，那么兼容。但是可能会丢失有效数字。当 XML 类型的值空间包含 -0 时，-0 在数据库中存储为 0。

12 兼容，插入的值是“0”（表示 false）或“1”（表示 true）。

13 如果处理完成后根据 db2-xdb:normalization 设置计算的 XML 输入字符串长度小于或等于目标列长度，那么兼容。而且，如果对此列映射将 db2-xdb:truncate 设置为“true”或“1”，也兼容。

13a 根据 13 中描述的条件确定是否兼容。此外，如果输入 XML 字符串的长度小于目标列的定义长度，那么插入该字符串时将在右边填充空白。

14 对于包含亚秒的 XML 值来说，仅当分解注释指定了值为“true”或“1”的 db2-xdb:truncate 时才兼容。对于带有时区指示符的 XML 值来说，如果 db2-xdb:truncate 设置为“true”或“1”，那么兼容；将插入不带时区的值。

15 如果年份为 4 位并且前面没有“-”符号，那么兼容。如果 XML 值不带时区指示符，那么兼容。如果 XML 值带有时区指示符，那么当 db2-xdb:truncate 设置为“true”或“1”时，值是兼容的。

16 如果值在 SQL 类型的范围内，但不是“INF”、“-INF”或“NaN”，那么兼容。当 XML 类型的值空间包含 -0 时，-0 在数据库中存储为 0。但是可能会丢失有效数字。

17 如果值不是“INF”、“-INF”或“NaN”，那么兼容。当 XML 类型的值空间包含 -0 时，-0 在数据库中存储为 0。

18 如果 URI 的字符串长度（以字节计）小于或等于目标列长度（以字节计），那么兼容。如果输入字符串比目标列长，那么仅当对此列映射将 db2-xdb:truncate 设置为“true”或“1”时，字符串才兼容。请注意，将插入 URI 本身（而不是 URI 指向的资源）。

18a 根据 18 中描述的条件确定是否兼容。此外，如果输入 XML 字符串的长度小于目标列的定义长度，那么插入该字符串时将在右边填充空白。

19 如果年份为 4 位并且前面没有“-”符号，那么兼容。对于带有时区指示符的 XML 值来说，如果 db2-xdb:truncate 设置为“true”或“1”，那么兼容。（在这种情况下，将插入不带时区的值。）如果指定了超过 6 位的亚秒，那么当 db2-xdb:truncate 设置为“true”或“1”时，值是兼容的。

20 如果年份为 4 位并且前面没有“-”符号，那么兼容。对于带有时区指示符的 XML 值来说，如果 db2-xdb:truncate 设置为“true”或“1”，那么兼容。（在这种情况下，将插入不带时区的日期值。）

21 数字的小数部分被截断。如果整数部分在 SQL 类型的范围内，那么兼容。当 XML 类型的值空间包含 -0 时，-0 在数据库中存储为 0。

22 数字的小数部分被截断。如果整数部分在 SQL 类型的范围内，并且值不是“INF”、“-INF”或“NaN”，那么兼容。当 XML 类型的值空间包含 -0 时，-0 在数据库中存储为 0。

带注释的 XML 模式分解的限制

带注释的 XML 模式分解存在特定限制。

限制

表 83. 带注释的 XML 模式分解的限制

条件	限制值
要分解的文档的最大大小	2 GB
单个带注释的 XML 模式中引用的最大表数	100
db2-xdb:expression 注释中的最大 $DECOMP_CONTENT 或 $DECOMP_ELEMENTID 实例数	10

表 83. 带注释的 XML 模式分解的限制 (续)

条件	限制值
db2-xdb:locationPath 中的最大层数	100
<xs:any> 或 <xs:anyAttribute> 的“namespace”属性中显式列示的最大名称空间数（如果该列表包含特殊值 ##targetNamespace 或 ##local，那么限制也适用于这些特殊值）	25
db2-xdb:name（表名）、db2-xdb:column、db2-xdb:defaultSQLSchema 或 db2-xdb:SQLSchema 值的最大字符串长度	与相应 DB2 对象的限制相同
db2-xdb:rowSet 值的最大字符串长度	与 db2-xdb:name 的限制相同
$DECOMP_CONTENT 的值的最大字符串长度	4096 字节

限制

带注释的 XML 模式分解不支持下列各项:

- 分解通配元素或属性: 不分解 XML 文档中与 XML 模式中的 <xs:any> 或 <xs:anyAttribute> 声明相对应的元素或属性。

  但是，如果这些元素或属性是在将 db2-xdb:contentHandling 设置为“serializeSubtree”或“stringValue”情况下分解的元素的子代，就会将通配元素或属性的内容作为序列化子树或字符串值的一部分进行分解。但是，这些通配元素或属性必须满足相应 <xs:any> 或 <xs:anyAttribute> 声明中指定的名称空间约束，这样才能对其进行序列化。
- 替换组: 对于替换组成员不仅仅用作文档的根元素的情况，如果 XML 文档包含替换组成员，并且 XML 模式包含组头，那么会发生错误。

  作为一种变通方法，可以将替换组的头和成员的元素声明更改为指定的 xs:choice 类型的模型组。例如，对于下列替换组声明:

```
<xs:element name="head" type="BaseType" />
<xs:element name="member1" type="derived1FromBaseType" substitutionGroup="head"/>
<xs:element name="member2" type="derived2FromBaseType" substitutionGroup="head"/>
<xs:element name="member3" type="derived3FromBaseType" substitutionGroup="head"/>
```

  can be changed to an equivalent named model group:

```
<xs:group name="mysubstitutiongrp">
  <xs:choice>
    <xs:element name="head" type="BaseType"/>
    <xs:element name="member1" type="derived1FromBaseType"/>
    <xs:element name="member2" type="derived2FromBaseType"/>
     <xs:element name="member3" type="derived3FromBaseType"/>
  </xs:choice>
</xs:group>
```

  可以将出现的 <head> 元素替换为 XML 文档中新定义的指定模型组。
- 使用 xsi:type 进行的运行时替换: 元素是根据与模式中元素名称相关联的模式类型中的映射来分解的。如果通过使用 xsi:type 来对文档中的元素指定另一类型，就会导致分解期间返回错误。

确保在 XML 文档中使用 xsi:type 指定的元素的类型与上下文中为该元素指定的类型匹配。如果不需要单独分解元素或者它的后代的内容，那么可以在 XML 模式中将该元素的类型更改为 xs:anyType。进行此更改之后，就不需要修改 XML 文档了。

- 递归元素：可在 XML 模式存储库（XSR）中注册包含递归的 XML 模式并对其启用分解。但是，关联 XML 实例文档的递归区域不能作为标量值分解到目标表中。通过使用相应的模式注释，可通过序列化标记的方式存储并检索递归部分。
- 更新或删除目标表中现有的行：分解功能只支持插入新行。（您仍可以在 XML 分解过程外部更新或删除行。）
- 从 NOTATION 派生的简单类型的属性：分解功能仅插入表示法名。
- ENTITY 类型的属性：分解功能仅插入实体名。
- 使用 db2-xdb:expression 和 db2-xdb:condition 多次映射至同一个行集和列：当多个项根据映射规则合法地映射至同一个行集和列时，那些映射不能包含 db2-xdb:expression 或 db2-xdb:condition 注释。

带注释的 XML 模式分解的故障诊断注意事项

如果您发现分解未产生期望的结果，应考虑某些事项。

一般注意事项

- 检查与 XML 模式相对应的 XSR 对象是否在 SYSCAT.XSROBJECTS 目录视图的 DECOMPOSITION 列中显示为已启用。如果 XSR 对象未启用，那么考虑执行禁用文档中描述的更正操作。
- 确保未超出 XML 分解的局限性与限制。
- 确保根据 XML 文档的 XML 模式，该 XML 文档有效。验证并不是分解的必需操作，但是，如果期望某种行为（如字符实体扩展），那么要执行带验证的分解。

XML 模式问题

- 确保 XML 模式未包含诸如内容模型不确定之类的错误，因为这些类型的错误会导致分解在执行验证时失败，或者在未执行验证的情况下生成未定义的分解结果。
- 确保仅对元素或属性声明（而未对复杂类型、元素/属性引用、模型组或任何其他 XML 模式结构）声明了非全局注释。并且，检查所声明注释的格式是否受支持：必须将注释声明为属性、元素或全局注释。（请参阅每个注释的文档以了解有关如何指定注释的详细信息。）
- 确保正确地对扩展或限制所派生的复杂类型添加注释。

特定错误

调整数据库配置参数可以解决下列错误：

- 当带注释的 XML 模式包含大量行集时，接收到 SQL0954：使用 applheapsz 配置参数增加应用程序堆大小
- 当带注释的 XML 模式的每个行集中包含复杂或许多表达式时，接收到 SQL0954：使用 applheapsz 配置参数增加应用程序堆大小
- 分解操作导致大量行时接收到 SQL0964：使用 **logprimary** 和 **logsecond** 配置参数增大可用的主日志文件或辅助日志文件数。还可以使用 **logfilsiz** 配置参数来增大主日志文件和辅助日志文件的大小。

锁定和并行性

如果在分解文档时遇到锁定升级或死锁，那么可通过应用程序调整并行控制。如果应用程序同时调用多个任意 xdbDecompXML 存储过程，其中多个分解操作中涉及许多相同的表，那么应用程序需要管理对这些表的并行访问，以防止出现锁定升级和死锁。

调整并行控制的一种方法是，在调用 xdbDecompXML 存储过程之前显式锁定分解中涉及的所有表。然后，在该存储过程返回后的适当时间执行 COMMIT 或 ROLLBACK 语句。因为分解大型文档可能导致插入许多行，并且由于缺省情况下在插入操作期间每个行被锁定，所以插入许多行的应用程序可能持有许多行锁定，从而导致锁定升级。通过获取表锁定，可以避免获取行锁定和锁定升级所产生的开销。

如果对您的应用程序来说，减少与获取表锁定相关联的并行性并不合适，那么可以增大 maxlocks 和/或 locklist 数据库配置参数，这将降低出现锁定升级的可能性。

设置 locktimeout 数据库配置参数以防止应用程序一直等待获取锁定。

映射目录视图中的验证

在验证以上情况后，如果仍遇到分解问题，请检查 SYSCAT.XDBMAPSHREDTREES 目录视图的 MAPPINGDESCRIPTION 列是否与所期望的映射匹配。MAPPINGDESCRIPTION 列包含有关如何映射行集中每个项的详细信息，这些详细信息包括:

- 目标列名
- 目标列类型
- 项的 XML 模式类型
- 对 db2-xdb:contentHandling、db2-xdb:normalization、db2-xdb:truncate、db2-xdb:expression 和 db2-xdb:condition 指定的值

请注意，SYSCAT.XDBMAPSHREDTREES 中除 MAPPINGDESCRIPTION 以外的列均用于 DB2 客户支持。

XML 分解注释的模式

带注释的 XML 模式分解支持一组分解注释，这些分解注释使您能够指定如何分解 XML 文档并将它们插入到数据库表中。本主题显示了 XML 分解定义的带注释模式的 XML 模式。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.ibm.com/xmlns/prod/db2/xdb1"
           targetNamespace="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" >
  <xs:element name="defaultSQLSchema" type="xs:string"/>
  <xs:attribute name="rowSet" type="xs:string"/>
  <xs:attribute name="column" type="xs:string"/>
  <xs:attribute name="locationPath" type="xs:string"/>
  <xs:attribute name="truncate" type="xs:boolean"/>
  <xs:attribute name="contentHandling">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="text"/>
        <xs:enumeration value="serializeSubtree"/>
        <xs:enumeration value="stringValue"/>
      </xs:restriction>
    </xs:simpleType>
```

```
  </xs:attribute>
  <xs:attribute name="normalization" >
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="original"/>
        <xs:enumeration value="whitespaceStrip"/>
        <xs:enumeration value="canonical"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="expression" type="xs:string"/>
  <xs:attribute name="condition" type="xs:string"/>
  <xs:element name="table">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SQLSchema" type="xs:string" minOccurs="0"/>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="rowSet" type="xs:string"
                    maxOccurs="unbounded" form="qualified"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="rowSetMapping">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="rowSet" type="xs:string" />
        <xs:element name="column" type="xs:string" minOccurs="0"/>
        <xs:element name="expression" type="xs:string" minOccurs="0" />
        <xs:element name="condition" type="xs:string" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute ref="truncate" />
      <xs:attribute ref="locationPath" />
      <xs:attribute ref="normalization" />
      <xs:attribute ref="contentHandling" />
    </xs:complexType>
  </xs:element>
  <xs:element name='rowSetOperationOrder'>
    <xs:complexType>
      <xs:choice minOccurs='1'  maxOccurs='1'>
        <xs:element name='order' type='orderType' minOccurs='1'
 maxOccurs='unbounded'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:complexType name='orderType'>
    <xs:sequence>
      <xs:element name='rowSet' type='xsd:string' minOccurs='2'
      maxOccurs='unbounded'/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

第 14 章 对 pureXML 的限制

对 pureXML 功能的限制

pureXML 功能存在某些限制，包括对 XML 列定义的限制、将表连接至分区表时的限制以及分区数据库环境中的限制。

对 XML 列定义的限制

XML 列存在以下限制:

- 对 XML 列创建索引时，必须使用 GENERATE KEY USING XMLPATTERN 子句，并且索引不能是组合索引的一部分。可对 XML 列创建多个索引。
- 只有在与 VALIDATED 谓词配合使用时才能在 CHECK 约束中引用 XML 列。
- XML 列不能具有 WITH DEFAULT 子句指定的缺省值；如果列可空，那么列的缺省值是 NULL
- 不能在范围集群表（RCT）中使用 XML 列。
- 不能将 XML 列包括为键列，此限制包括主键、外键和唯一键、多维集群（MDC）表的维键（在 ORGANIZE BY 子句中）、范围集群表的序列键、分区表的表分区键以及分区数据库环境中的表的分布键。
- 带有 XML 列的分区表必须至少有一个非 XML 列，其数据类型支持用作表分区键列。
- XML 列不能包括在类型表和带类型视图中。
- 不能在生成列中引用 XML 列。
- 不能在可滚动游标的选择列表中指定 XML 列。
- XML 列会导致检索 XML 数据时禁用游标分块。
- 使用 ALTER TABLE 语句删除 XML 列时，必须删除单个 ALTER TABLE 语句中的表的所有 XML 列。

对触发器的限制

在 BEFORE 或 AFTER 触发器的触发器主体中，引用受影响行中类型为 XML 的列的转换变量只能与 XMLVALIDATE 函数配合使用进行验证，以将 XML 列值设置为 NULL，或保留 XML 列值不变。

将分区连接至分区表时的限制

使用 ALTER ATTACH 将分区连接到带有 XML 列的分区表时，要连接的表（源表）上每个 XML 列的 INLINE LENGTH 必须与连接至的表（目标表）上对应 XML 列的 INLINE LENGTH 匹配。

如果表包含的 XML 列使用版本 9.5 或之前版本的 XML 记录格式，那么不支持将该表连接至包含以下 XML 列的分区表：这些列使用版本 9.7 或更高版本记录格式。在连接至该表之前，必须更新该表的 XML 记录格式以便与目标分区表的记录格式相匹配。下列两种方法都可用于更新表的 XML 记录格式:

- 使用 ADMIN_MOVE_TABLE 过程来对表执行联机表移动。

- 执行下列步骤:
  1. 使用 EXPORT 命令来创建表数据的副本。
  2. 使用 TRUNCATE 语句来删除该表中的所有行并释放分配给该表的存储空间。
  3. 使用 LOAD 命令将该数据添加到表中。

更新该表的 XML 记录格式后，将该表连接至目标分区表。

分区数据库环境中的限制

在分区数据库环境中使用 pureXML 功能时，存在以下规则:

- 不能将 XML 列用作分布键。因此，存在以下限制:
  – 不能分发仅包含 XML 列的表。
  – 带有分布键的表不能有主键、唯一约束或对 XML 列定义的唯一索引。
  – 带有分布键和 XML 列的表必须至少具有一个非 XML 列，其数据类型支持用作分布键。
- 如果要以并行方式将 XML 数据文件中的 XML 数据装入到分区表中，那么正在进行并行装入的所有分区必须能够对 XML 数据文件执行读访问。
- 将 CURSOR 文件类型与 LOAD 命令配合使用将 XML 数据装入到多分区数据库中时，不支持 PARTITION_ONLY 和 LOAD_ONLY 方式。
- 将 REDISTRIBUTE DATABASE PARTITION GROUP 命令与 NOT ROLLFORWARD RECOVERABLE 选项配合使用时，再分发操作将对包含 XML 列的表使用 INDEXING MODE DEFERRED 选项。如果表未包含 XML 列，那么再分发操作将使用在发出该命令时所指定的建立索引方式。
- 如果表包含的 XML 列使用版本 9.5 或之前版本的 XML 记录格式，那么不能对这些表执行再分发操作。使用 ADMIN_MOVE_TABLE 存储过程将该表迁移至新格式。

对数据行压缩的限制

ALTER TABLE 或 CREATE TABLE 语句的 COMPRESS YES 选项允许对表进行数据行压缩。如果表包含的 XML 列使用版本 9.5 或之前版本的 XML 记录格式，那么不支持对表的 XML 存储对象执行数据压缩。如果对这样的表启用数据行压缩，那么将只压缩表对象中的表行数据。

如果允许对表进行数据行压缩，但在插入、装入或重组操作期间不能对该表的 XML 存储对象进行压缩，那么会在 db2diag 日志文件中写入一个消息。

要使表的 XML 存储对象中的数据符合压缩条件，请使用 ADMIN_MOVE_TABLE 存储过程来迁移此表，然后对已迁移的表启用数据行压缩。

其他限制

虽然对存储在数据库中的 XML 值没有体系结构限制，但与数据库交换的序列化 XML 数据被有效地限制为 2GB。

如果完成 XML 数据的 ALLOW READ ACCESS 装入操作并且让表保留集合完整性暂挂状态，那么可对此表发出 RUNSTATS 命令。在此方案中，RUNSTATS 操作看不到上一装入操作中不可视的 XML 索引键并返回错误。变通方法是在 RUNSTATS 命令之前返回 SET INTEGRITY 语句。

使用 LOAD 命令装入 XML 数据时，在下列情况下不支持使用 FOR EXCEPTION 子句来指定装入异常表:

- 使用基于标签的访问控制（LBAC）时。
- 将数据装入到分区表中时。

对 XML 列创建索引和使用 XSLT 样式表进行变换时，存在其他限制。

附录 A. 编码映射

将编码名映射至已存储的 XML 数据的有效 CCSID

如果 XML 列中存储的数据包含在二进制应用程序变量中，或者该数据是内部编码的 XML 类型，那么 DB2 数据库管理器就会检查该数据以确定编码。如果该数据包含编码声明，数据库管理器就会将编码名称映射至 CCSID。

第 284 页的表 49 列示了这些映射。如果编码名未包含在第 284 页的表 49 中，数据库管理器就会返回错误。

第 284 页的表 49 第一列中的规范化编码名是通过将编码名转换为大写并除去所有连字符、加号、下划线、冒号、句点和空格得到。例如，ISO88591 是 ISO 8859-1、ISO-8859-1 和 iso-8859-1 的规范化编码名。

表 84. 编码名和有效 CCSID

规范化编码名	CCSID
437	437
646	367
813	813
819	819
850	850
852	852
855	855
857	857
862	862
863	863
866	866
869	869
885913	901
885915	923
88591	819
88592	912
88595	915
88597	813
88598	62210
88599	920
904	904
912	912
915	915
916	916
920	920

表 84. 编码名和有效 CCSID（续）

规范化编码名	CCSID
923	923
ANSI1251	1251
ANSIX341968	367
ANSIX341986	367
ARABIC	1089
ASCII7	367
ASCII	367
ASMO708	1089
BIG5	950
CCSID00858	858
CCSID00924	924
CCSID01140	1140
CCSID01141	1141
CCSID01142	1142
CCSID01143	1143
CCSID01144	1144
CCSID01145	1145
CCSID01146	1146
CCSID01147	1147
CCSID01148	1148
CCSID01149	1149
CP00858	858
CP00924	924
CP01140	1140
CP01141	1141
CP01142	1142
CP01143	1143
CP01144	1144
CP01145	1145
CP01146	1146
CP01147	1147
CP01148	1148
CP01149	1149
CP037	37
CP1026	1026
CP1140	1140
CP1141	1141
CP1142	1142
CP1143	1143
CP1144	1144

表 84. 编码名和有效 CCSID（续）

规范化编码名	CCSID
CP1145	1145
CP1146	1146
CP1147	1147
CP1148	1148
CP1149	1149
CP1250	1250
CP1251	1251
CP1252	1252
CP1253	1253
CP1254	1254
CP1255	1255
CP1256	1256
CP1257	1257
CP1258	1258
CP1363	1363
CP1383	1383
CP1386	1386
CP273	273
CP277	277
CP278	278
CP280	280
CP284	284
CP285	285
CP297	297
CP33722	954
CP33722C	954
CP367	367
CP420	420
CP423	423
CP424	424
CP437	437
CP500	500
CP5346	5346
CP5347	5347
CP5348	5348
CP5349	5349
CP5350	5350
CP5353	5353
CP813	813
CP819	819

表 84. 编码名和有效 CCSID（续）

规范化编码名	CCSID
CP838	838
CP850	850
CP852	852
CP855	855
CP857	857
CP858	858
CP862	862
CP863	863
CP864	864
CP866	866
CP869	869
CP870	870
CP871	871
CP874	874
CP904	904
CP912	912
CP915	915
CP916	916
CP920	920
CP921	921
CP922	922
CP923	923
CP936	1386
CP943	943
CP943C	943
CP949	970
CP950	950
CP964	964
CP970	970
CPGR	869
CSASCII	367
CSBIG5	950
CSEBCDICCAFR	500
CSEBCDICDKNO	277
CSEBCDICES	284
CSEBCDICFISE	278
CSEBCDICFR	297
CSEBCDICIT	280
CSEBCDICPT	37
CSEBCDICUK	285

表 84. 编码名和有效 CCSID（续）

规范化编码名	CCSID
CSEBCDICUS	37
CSEUCKR	970
CSEUCPKDFMTJAPANESE	954
CSGB2312	1383
CSHPROMAN8	1051
CSIBM037	37
CSIBM1026	1026
CSIBM273	273
CSIBM277	277
CSIBM278	278
CSIBM280	280
CSIBM284	284
CSIBM285	285
CSIBM297	297
CSIBM420	420
CSIBM423	423
CSIBM424	424
CSIBM500	500
CSIBM855	855
CSIBM857	857
CSIBM863	863
CSIBM864	864
CSIBM866	866
CSIBM869	869
CSIBM870	870
CSIBM871	871
CSIBM904	904
CSIBMEBCDICATDE	273
CSIBMTHAI	838
CSISO128T101G2	920
CSISO146SERBIAN	915
CSISO147MACEDONIAN	915
CSISO2INTLREFVERSION	367
CSISO646BASIC1983	367
CSISO88596I	1089
CSISO88598I	916
CSISOLATIN0	923
CSISOLATIN1	819
CSISOLATIN2	912
CSISOLATIN5	920

表 84. 编码名和有效 CCSID（续）

规范化编码名	CCSID
CSISOLATIN9	923
CSISOLATINARABIC	1089
CSISOLATINCYRILLIC	915
CSISOLATINGREEK	813
CSISOLATINHEBREW	62210
CSKOI8R	878
CSKSC56011987	970
CSMACINTOSH	1275
CSMICROSOFTPUBLISHING	1004
CSPC850MULTILINGUAL	850
CSPC862LATINHEBREW	862
CSPC8CODEPAGE437	437
CSPCP852	852
CSSHIFTJIS	943
CSUCS4	1236
CSUNICODE11	1204
CSUNICODE	1204
CSUNICODEASCII	1204
CSUNICODELATIN1	1204
CSVISCII	1129
CSWINDOWS31J	943
CYRILLIC	915
DEFAULT	367
EBCDICATDE	273
EBCDICCAFR	500
EBCDICCPAR1	420
EBCDICCPBE	500
EBCDICCPCA	37
EBCDICCPCH	500
EBCDICCPDK	277
EBCDICCPES	284
EBCDICCPFI	278
EBCDICCPFR	297
EBCDICCPGB	285
EBCDICCPGR	423
EBCDICCPHE	424
EBCDICCPIS	871
EBCDICCPIT	280
EBCDICCPNL	37
EBCDICCPNO	277

表 84. 编码名和有效 CCSID (续)

规范化编码名	CCSID
EBCDICCPROECE	870
EBCDICCPSE	278
EBCDICCPUS	37
EBCDICCPWT	37
EBCDICCPYU	870
EBCDICDE273EURO	1141
EBCDICDK277EURO	1142
EBCDICDKNO	277
EBCDICES284EURO	1145
EBCDICES	284
EBCDICFI278EURO	1143
EBCDICFISE	278
EBCDICFR297EURO	1147
EBCDICFR	297
EBCDICGB285EURO	1146
EBCDICINTERNATIONAL500EURO	1148
EBCDICIS871EURO	1149
EBCDICIT280EURO	1144
EBCDICIT	280
EBCDICLATIN9EURO	924
EBCDICNO277EURO	1142
EBCDICPT	37
EBCDICSE278EURO	1143
EBCDICUK	285
EBCDICUS37EURO	1140
EBCDICUS	37
ECMA114	1089
ECMA118	813
ELOT928	813
EUCCN	1383
EUCJP	954
EUCKR	970
EUCTW	964
EXTENDEDUNIXCODEPACKEDFORMATFORJAPANESE	954
GB18030	1392
GB2312	1383
GBK	1386
GREEK8	813
GREEK	813
HEBREW	62210

表 84. 编码名和有效 CCSID（续）

规范化编码名	CCSID
HPROMAN8	1051
IBM00858	858
IBM00924	924
IBM01140	1140
IBM01141	1141
IBM01142	1142
IBM01143	1143
IBM01144	1144
IBM01145	1145
IBM01146	1146
IBM01147	1147
IBM01148	1148
IBM01149	1149
IBM01153	1153
IBM01155	1155
IBM01160	1160
IBM037	37
IBM1026	1026
IBM1043	1043
IBM1047	1047
IBM1252	1252
IBM273	273
IBM277	277
IBM278	278
IBM280	280
IBM284	284
IBM285	285
IBM297	297
IBM367	367
IBM420	420
IBM423	423
IBM424	424
IBM437	437
IBM500	500
IBM808	808
IBM813	813
IBM819	819
IBM850	850
IBM852	852
IBM855	855

表 84. 编码名和有效 CCSID（续）

规范化编码名	CCSID
IBM857	857
IBM862	862
IBM863	863
IBM864	864
IBM866	866
IBM867	867
IBM869	869
IBM870	870
IBM871	871
IBM872	872
IBM902	902
IBM904	904
IBM912	912
IBM915	915
IBM916	916
IBM920	920
IBM921	921
IBM922	922
IBM923	923
IBMTHAI	838
IRV	367
ISO10646	1204
ISO10646UCS2	1200
ISO10646UCS4	1232
ISO10646UCSBASIC	1204
ISO10646UNICODELATIN1	1204
ISO646BASIC1983	367
ISO646IRV1983	367
ISO646IRV1991	367
ISO646US	367
ISO885911987	819
ISO885913	901
ISO885915	923
ISO885915FDIS	923
ISO88591	819
ISO885921987	912
ISO88592	912
ISO885951988	915
ISO88595	915
ISO885961987	1089

表 84. 编码名和有效 CCSID（续）

规范化编码名	CCSID
ISO88596	1089
ISO88596I	1089
ISO885971987	813
ISO88597	813
ISO885981988	62210
ISO88598	62210
ISO88598I	916
ISO885991989	920
ISO88599	920
ISOIR100	819
ISOIR101	912
ISOIR126	813
ISOIR127	1089
ISOIR128	920
ISOIR138	62210
ISOIR144	915
ISOIR146	915
ISOIR147	915
ISOIR148	920
ISOIR149	970
ISOIR2	367
ISOIR6	367
JUSIB1003MAC	915
JUSIB1003SERB	915
KOI8	878
KOI8R	878
KOI8U	1168
KOREAN	970
KSC56011987	970
KSC56011989	970
KSC5601	970
L1	819
L2	912
L5	920
L9	923
LATIN0	923
LATIN1	819
LATIN2	912
LATIN5	920
LATIN9	923

表 84. 编码名和有效 CCSID（续）

规范化编码名	CCSID
MAC	1275
MACEDONIAN	915
MACINTOSH	1275
MICROSOFTPUBLISHING	1004
MS1386	1386
MS932	943
MS936	1386
MS949	970
MSKANJI	943
PCMULTILINGUAL850EURO	858
R8	1051
REF	367
ROMAN8	1051
SERBIAN	915
SHIFTJIS	943
SJIS	943
SUNEUGREEK	813
T101G2	920
TIS20	874
TIS620	874
UNICODE11	1204
UNICODE11UTF8	1208
UNICODEBIGUNMARKED	1200
UNICODELITTLEUNMARKED	1202
US	367
USASCII	367
UTF16	1204
UTF16BE	1200
UTF16LE	1202
UTF32	1236
UTF32BE	1232
UTF32LE	1234
UTF8	1208
VISCII	1129
WINDOWS1250	1250
WINDOWS1251	1251
WINDOWS1252	1252
WINDOWS1253	1253
WINDOWS1254	1254
WINDOWS1255	1255

表 84. 编码名和有效 CCSID (续)

规范化编码名	CCSID
WINDOWS1256	1256
WINDOWS1257	1257
WINDOWS1258	1258
WINDOWS28598	62210
WINDOWS31J	943
WINDOWS936	1386
XEUCTW	964
XMSWIN936	1386
XUTF16BE	1200
XUTF16LE	1202
XWINDOWS949	970

将 CCSID 映射至序列化 XML 输出数据的编码名

作为隐式或显式 XMLSERIALIZE 操作的一部分，DB2 数据库管理器可能会在序列化 XML 输出数据开头添加编码声明。

该声明的格式如下所示:

```
<?xml version="1.0" encoding="encoding-name"?>
```

通常，编码声明中的字符集标识描述输出字符串中的字符编码。例如，在将 XML 数据序列化为与目标应用程序数据类型对应的 CCSID 时，编码声明将描述目标应用程序变量 CCSID。应用程序执行指定了 INCLUDING XMLDECLARATION 的显式 XMLSERIALIZE 函数时的情况例外。如果指定了 INCLUDING XMLDECLARATION，数据库管理器就会为 UTF-8 生成编码声明。如果目标数据类型是 CLOB 或 DBCLOB 类型，就可能会执行其他代码页转换操作，这会导致编码信息不准确。如果在应用程序中对该数据进行进一步解析，就可能会导致数据损坏。

根据 XML 标准的规定，DB2 数据库管理器尽可能地选择 CCSID 的 IANA 注册表名。

表 85. CCSID 和对应的编码名

CCSID	编码名
37	IBM037
273	IBM273
277	IBM277
278	IBM278
280	IBM280
284	IBM284
285	IBM285
297	IBM297
367	US-ASCII
420	IBM420
423	IBM423

表 85. CCSID 和对应的编码名 (续)

CCSID	编码名
424	IBM424
437	IBM437
500	IBM500
808	IBM808
813	ISO-8859-7
819	ISO-8859-1
838	IBM-Thai
850	IBM850
852	IBM852
855	IBM855
857	IBM857
858	IBM00858
862	IBM862
863	IBM863
864	IBM864
866	IBM866
867	IBM867
869	IBM869
870	IBM870
871	IBM871
872	IBM872
874	TIS-620
878	KOI8-R
901	ISO-8859-13
902	IBM902
904	IBM904
912	ISO-8859-2
915	ISO-8859-5
916	ISO-8859-8-I
920	ISO-8859-9
921	IBM921
922	IBM922
923	ISO-8859-15
924	IBM00924
932	Shift_JIS
943	Windows-31J
949	EUC-KR
950	Big5
954	EUC-JP
964	EUC-TW

表 85. CCSID 和对应的编码名 (续)

CCSID	编码名
970	EUC-KR
1004	Microsoft-Publish
1026	IBM1026
1043	IBM1043
1047	IBM1047
1051	hp-roman8
1089	ISO-8859-6
1129	VISCII
1140	IBM01140
1141	IBM01141
1142	IBM01142
1143	IBM01143
1144	IBM01144
1145	IBM01145
1146	IBM01146
1147	IBM01147
1148	IBM01148
1149	IBM01149
1153	IBM01153
1155	IBM01155
1160	IBM-Thai
1161	TIS-620
1162	TIS-620
1163	VISCII
1168	KOI8-U
1200	UTF-16BE
1202	UTF-16LE
1204	UTF-16
1208	UTF-8
1232	UTF-32BE
1234	UTF-32LE
1236	UTF-32
1250	windows-1250
1251	windows-1251
1252	windows-1252
1253	windows-1253
1254	windows-1254
1255	windows-1255
1256	windows-1256
1257	windows-1257

表 85. CCSID 和对应的编码名 (续)

CCSID	编码名
1258	windows-1258
1275	MACINTOSH
1363	KSC_5601
1370	Big5
1381	GB2312
1383	GB2312
1386	GBK
1392	GB18030
4909	ISO-8859-7
5039	Shift_JIS
5346	windows-1250
5347	windows-1251
5348	windows-1252
5349	windows-1253
5350	windows-1254
5351	windows-1255
5352	windows-1256
5353	windows-1257
5354	windows-1258
5488	GB18030
8612	IBM420
8616	IBM424
9005	ISO-8859-7
12712	IBM424
13488	UTF-16BE
13490	UTF-16LE
16840	IBM420
17248	IBM864
17584	UTF-16BE
17586	UTF-16LE
62209	IBM862
62210	ISO-8859-8
62211	IBM424
62213	IBM862
62215	ISO-8859-8
62218	IBM864
62221	IBM862
62222	ISO-8859-8
62223	windows-1255
62224	IBM420

表 85. CCSID 和对应的编码名 (续)

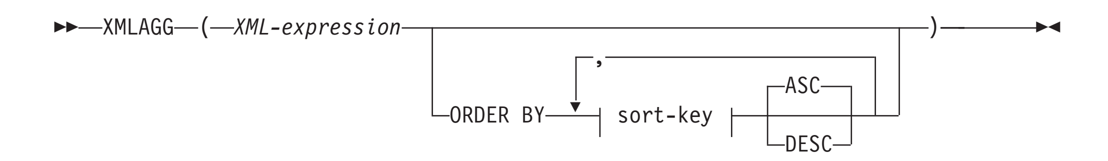
CCSID	编码名
62225	IBM864
62227	ISO-8859-6
62228	windows-1256
62229	IBM424
62231	IBM862
62232	ISO-8859-8
62233	IBM420
62234	IBM420
62235	IBM424
62237	windows-1255
62238	ISO-8859-8-I
62239	windows-1255
62240	IBM424
62242	IBM862
62243	ISO-8859-8-I
62244	windows-1255
62245	IBM424
62250	IBM420

附录 B. SQL/XML 发布函数

以下各节描述 DB2 SQL/XML 发布函数的语法。

有关使用这些函数的信息，请参阅第 109 页的『用于构造 XML 值的 SQL/XML 发布函数』。

XMLAGG



模式为 SYSIBM。不能将函数名指定为限定名。

XMLAGG 函数将返回一个 XML 序列，对于 XML 值集合中的每个非空值，该序列都包含一项。

XML-expression
指定数据类型为 XML 的表达式。

ORDER BY
指定聚集中处理的同一分组集合中的行顺序。如果省略了 ORDER BY 子句，或者 ORDER BY 子句不能区分列数据的顺序，那么同一分组集合中的行将任意排序。

sort-key
排序键可以是列名或 *sort-key-expression*。注意，如果排序键为常量，那么它不会像普通 ORDER BY 子句中那样引用输出列的位置，但它只是一个常量，这意味着没有排序键。

结果的数据类型为 XML。

该函数将应用于因为消除空值而从自变量值中派生出来的值集合。

如果 *XML-expression* 自变量可为空，那么结果可为空。如果值集合是空的，那么结果为空。否则，结果为 XML 序列，对于集合中的每个值，该序列都包含一项。

注:

1. **在 OLAP 表达式中的支持**: XMLAGG 不能用作 OLAP 聚集函数的列函数（SQLSTATE 42601）。

示例:

注: XMLAGG 不会在输出中插入空格或换行符。所有示例输出都将格式化以增强可读性。

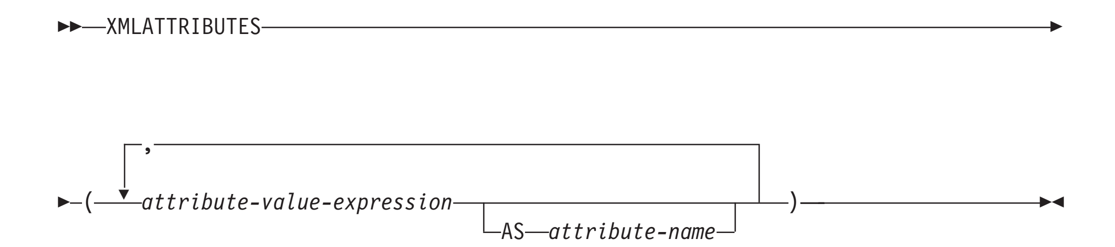
- 为每个部门构造部门元素，其中包含按姓氏排序的职员列表。


```
SELECT XMLSERIALIZE(
  CONTENT XMLELEMENT(
    NAME "Department", XMLATTRIBUTES(
      E.WORKDEPT AS "name"
    ),
    XMLAGG(
      XMLELEMENT(
        NAME "emp", E.LASTNAME
      )
      ORDER BY E.LASTNAME
    )
  )
  AS CLOB(110)
)
AS "dept_list"
FROM EMPLOYEE E
WHERE E.WORKDEPT IN ('C01','E21')
GROUP BY WORKDEPT
```

此查询将生成以下结果:

```
dept_list
-----------------------...
<Department name="C01">
  <emp>KWAN</emp>
  <emp>NICHOLLS</emp>
  <emp>QUINTANA</emp>
</Department>
<Department name="E21">
  <emp>GOUNOT</emp>
  <emp>LEE</emp>
  <emp>MEHTA</emp>
  <emp>SPENSER</emp>
</Department>
```

XMLATTRIBUTES



模式为 SYSIBM。不能将函数名指定为限定名。

XMLATTRIBUTES 函数将通过自变量构造 XML 属性。此函数只能用作 XMLELEMENT 函数的自变量。结果是一个 XML 序列，对于每个非空输入值，该序列都包含一个 XQuery 属性节点。

attribute-value-expression
: 结果为属性值的表达式。*attribute-value-expression* 的数据类型不能是结构化类型（SQLSTATE 42884）。该表达式可以是任何 SQL 表达式。如果该表达式不是简单列引用，那么必须指定属性名称。

attribute-name
: 指定属性名称。该名称是必须以 XML 限定名称或 QName（SQLSTATE 42634）格式出现的 SQL 标识。有关有效名称的更多详细信息，请参阅 W3C XML 名称空

间规范。该属性名称不能是 xmlns 或以 xmlns: 为前缀。名称空间是使用函数 XMLNAMESPACES 声明的。不管是隐式还是显式，都不允许使用重复的属性名称（SQLSTATE 42713）。

如果未指定 *attribute-name*，那么 *attribute-value-expression* 必须是列名（SQLSTATE 42703）。属性名称是根据列名使用从列名至 XML 属性名称的完全转义映射创建的。

结果的数据类型为 XML。如果 *attribute-value-expression* 可为空，那么结果可为空；如果每个 *attribute-value-expression* 为空，那么结果为空。

示例:

注: XMLATTRIBUTES 不会在输出中插入空格或换行符。所有示例输出都将格式化以增强可读性。

- 将使用属性生成元素。

```
SELECT E.EMPNO, XMLELEMENT(
  NAME "Emp",
  XMLATTRIBUTES(
    E.EMPNO, E.FIRSTNME ||' '|| E.LASTNAME AS "name"
  )
)
AS "Result"
FROM EMPLOYEE E WHERE E.EDLEVEL = 12
```

  此查询将生成以下结果:

```
EMPNO  Result
000290 <Emp EMPNO="000290" name="JOHN PARKER"></Emp>
000310 <Emp EMPNO="000310" name="MAUDE SETRIGHT"></Emp>
200310 <Emp EMPNO="200310" name="MICHELLE SPRINGER"></Emp>
```

- 使用任何 QName 中未使用的名称空间声明来生成元素。将在属性值中使用前缀。

```
VALUES XMLELEMENT(
  NAME "size",
  XMLNAMESPACES(
    'http://www.w3.org/2001/XMLSchema-instance' AS "xsi",
    'http://www.w3.org/2001/XMLSchema' AS "xsd"
  ),
  XMLATTRIBUTES(
    'xsd:string' AS "xsi:type"
  ), '1'
)
```

  此查询将生成以下结果:

```
<size xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xsi:type="xsd:string">1</size>
```

XMLCOMMENT

```
►►──XMLCOMMENT──(──string-expression──)──────────────────────────►◄
```

模式为 SYSIBM。不能将函数名指定为限定名。

XMLCOMMENT 函数将返回具有单个 XQuery 注释节点的 XML 值，该注释节点将输入自变量作为内容。

string-expression

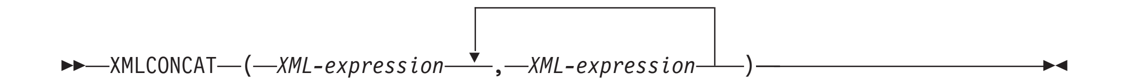
其值为字符串类型 CHAR、VARCHAR 或 CLOB 的表达式。将解析 *string-expression* 的结果以针对 XML 1.0 规则中指定的内容检查它是否符合 XML 注释的要求。*string-expression* 的结果必须符合以下正则表达式:

```
((Char - '-') | ('-' (Char - '-')))*
```

其中 Char 被定义为超大字符集块 X'FFFE' 和 X'FFFF' 以外的任何 Unicode 字符。基本上 XML 注释不能包含两个相邻连字符，也不能以连字符结尾（SQLSTATE 2200S）。

结果的数据类型为 XML。如果 *string-expression* 的结果可为空，那么结果可为空；如果输入值为空，那么结果为空。

XMLCONCAT



模式为 SYSIBM。不能将函数名指定为限定名。

XMLCONCAT 函数将返回一个序列，该序列包含数目不定的 XML 输入自变量的并置。

XML-expression

指定数据类型为 XML 的表达式。

结果的数据类型为 XML。结果是一个 XML 序列，该序列包含非空输入 XML 值的并置。输入中的空值将被忽略。如果任何 *XML-expression* 的结果可为空，那么结果可为空；如果每个输入值的结果为空，那么结果为空。

示例:

注: XMLCONCAT 不会在输出中插入空格或换行符。所有示例输出都将格式化以增强可读性。

- 为部门 A00 和 B01 构造部门元素，其中包含按名字排序的职员列表。在部门元素前加上介绍性注释。

```
SELECT XMLCONCAT(
  XMLCOMMENT(
    'Confirm these employees are on track for their product schedule'
  ),
  XMLELEMENT(
    NAME "Department",
    XMLATTRIBUTES(
      E.WORKDEPT AS "name"
    ),
    XMLAGG(
      XMLELEMENT(
        NAME "emp", E.FIRSTNME
      )
    ORDER BY E.FIRSTNME
    )
  )
```

```
)
FROM EMPLOYEE E
WHERE E.WORKDEPT IN ('A00', 'B01')
GROUP BY E.WORKDEPT
```

此查询将生成以下结果:

```
<!--Confirm these employees are on track for their product schedule-->
<Department name="A00">
<emp>CHRISTINE</emp>
<emp>DIAN</emp>
<emp>GREG</emp>
<emp>SEAN</emp>
<emp>VINCENZO</emp>
</Department>
<!--Confirm these employees are on track for their product schedule-->
<Department name="B01">
<emp>MICHAEL</emp>
</Department>
```

XMLDOCUMENT

```
►►—XMLDOCUMENT—(—XML-expression—)——————————————————————►◄
```

模式为 SYSIBM。不能将函数名指定为限定名。

XMLDOCUMENT 函数返回具有单个 XQuery 文档节点的 XML 值，该文档节点具有零个或多个子节点。

XML-expression
: 返回 XML 值的表达式。XML 值的序列项一定不能是属性节点（SQLSTATE 10507）。

结果的数据类型为 XML。如果任何 *XML-expression* 的结果可为空，那么结果可为空；如果输入值为空，那么结果为空。

生成文档节点的子代将按下列步骤中描述的方式构造。输入表达式是节点或原子值序列，在这些步骤中又称为内容序列。

1. 如果内容序列包含文档节点，那么该文档节点在内容序列中将替换为文档节点的子代。
2. 内容序列中由一个或多个原子值组成的每个相邻序列将替换为文本节点，该文本节点包含将每个原子值强制转换为字符串的结果（相邻值之间插入了单个空白字符）。
3. 对于内容序列中的每个节点，将为该节点构造新的深层副本。节点的深层副本是根植于该节点的整个子树（包括节点本身及其后代）的副本。每个被复制节点具有新的节点标识。
4. 内容序列中出现的节点将成为新文档节点的子代。

XMLDOCUMENT 函数将有效执行 XQuery 计算文档构造函数。

```
XMLQUERY('document {$E}' PASSING BY REF XML-expression AS "E")
```

的结果相当于

```
XMLDOCUMENT( XML-expression )
```

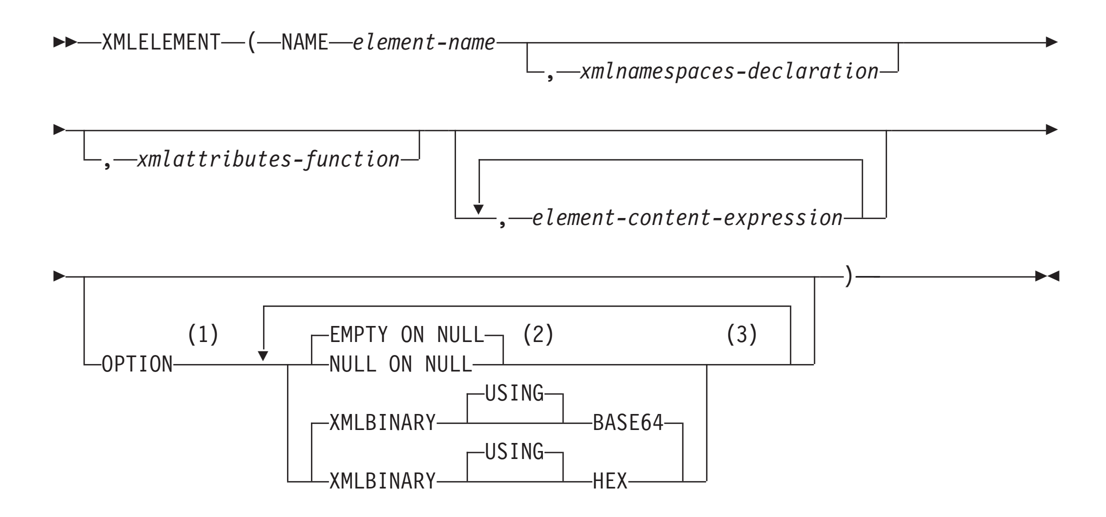
，*XML-expression* 为空并且 XMLQUERY 相对于 XMLDOCUMENT 返回空序列（XMLDOCUMENT 返回空值）的情况下例外。

示例:

- 在 XML 列中插入已构造文档。

```
INSERT INTO T1 VALUES(
  123, (
    SELECT XMLDOCUMENT(
      XMLELEMENT(
        NAME "Emp", E.FIRSTNME || ' ' || E.LASTNAME, XMLCOMMENT(
          'This is just a simple example'
        )
      )
    )
    FROM EMPLOYEE E
    WHERE E.EMPNO = '000120'
  )
)
```

XMLELEMENT



注:

1 仅当指定至少一个 *xmlattributes-function* 或 *element-content-expression* 后，才能指定 OPTION 子句。

2 仅当指定至少一个 *element-content-expression* 时，才能指定 NULL ON NULL 或 EMPTY ON NULL。

3 不能多次指定同一子句。

模式为 SYSIBM。不能将函数名指定为限定名。

XMLELEMENT 函数返回的 XML 值是 XQuery 元素节点。

NAME *element-name*
指定 XML 元素的名称。该名称是必须以 XML 限定名称或 QName（SQLSTATE 42634）格式出现的 SQL 标识。有关有效名称的更多详细信息，请参阅 W3C XML 名称空间规范。如果是限定名称，那么必须在作用域中声明名称空间前缀（SQLSTATE 42635）。

xmlnamespaces-declaration

指定 XMLNAMESPACES 声明生成的 XML 名称空间声明。声明的名称空间在 XMLELEMENT 函数作用域中。不管是否出现在另一子查询中，这些名称空间都适用于 XMLELEMENT 函数内的任何嵌套 XML 函数。

如果未指定 *xmlnamespaces-declaration*，那么名称空间声明与已构造元素无关。

xmlattributes-function

指定元素的 XML 属性。这些属性是 XMLATTRIBUTES 函数生成的。

element-content-expression

生成的 XML 元素节点的内容是由一个表达式或一列表达式指定的。*element-content-expression* 的数据类型不能是结构化类型（SQLSTATE 42884）。该表达式可以是任何 SQL 表达式。

如果未指定 *element-content-expression*，那么空字符串将用作元素的内容并且一定不能指定 OPTION NULL ON NULL 或 EMPTY ON NULL。

OPTION

指定用于构造 XML 元素的其他选项。如果未指定 OPTION 子句，那么缺省值为 EMPTY ON NULL XMLBINARY USING BASE64。此子句对 *element-content-expression* 中指定的嵌套 XMLELEMENT 调用没有影响。

EMPTY ON NULL 或 NULL ON NULL

如果每个 *element-content-expression* 的值都是空值，那么指定是返回空值还是空元素。此选项仅影响元素内容的空值处理，而不会影响属性值。缺省值为 EMPTY ON NULL。

EMPTY ON NULL

如果每个 *element-content-expression* 的值都为空，那么会返回空元素。

NULL ON NULL

如果每个 *element-content-expression* 的值为空，那么会返回空值。

XMLBINARY USING BASE64 或 XMLBINARY USING HEX

指定二进制输入数据的采用编码、带有 FOR BIT DATA 属性的字符串或基于其中一种类型的单值类型。该编码适用于元素内容或属性值。缺省值为 XMLBINARY USING BASE64。

XMLBINARY USING BASE64

根据对 XML 模式类型 xs:base64Binary 编码的定义，指定采用编码为基本 64 位字符。基本 64 位代码使用 US-ASCII 的由 65 个字符组成的子集（10 个数字、26 个小写字符、26个大写字符以及“+”和“ ”）来表示每个二进制数据或位数据的 6 位以及子集中的一个可打印字符。这些字符已被选中，所以它们是以可通用的方式表示的。使用此方法时，编码数据的大小比原始二进制数据或位数据大 33%。

XMLBINARY USING HEX

根据对 XML 模式类型 xs:hexBinary 编码的定义，指定采用编码为十六进制字符。十六进制编码使用两个十六进制字符来表示每个字节（8 位）。使用此方法时，编码数据的大小是原始二进制数据或位数据的两倍。

此函数使用元素名称、名称空间声明的可选集合、属性的可选集合以及零个或多个自变量来构成 XML 元素的内容。结果是包含 XML 元素或空值的 XML 序列。

结果的数据类型为 XML。如果任何 *element-content-expression* 自变量可为空，那么结果可为空；如果所有 *element-content-expression* 自变量值为空并且 NULL ON NULL 选项生效，那么结果为空。

注:

1. 构造将作为另一元素（用于定义缺省名称空间）的内容进行复制的元素时，应在被复制元素中显式取消声明缺省名称空间，以避免因为从新的父代元素继承缺省名称空间而可能导致的错误。预定义名称空间前缀（“xs”、“xsi”、“xml”和“sqlxml”）在使用时也必须显式声明。
2. **构造元素节点**: 生成的元素节点按如下方式构造:
   a. *xmlnamespaces-declaration* 将对已构造元素添加一组名称空间作用域。每个名称空间作用域会将名称空间前缀（或缺省名称空间）与名称空间 URI 相关联。名称空间作用域将定义一组名称空间前缀，这些前缀可用于解释元素作用域内的 QNames。
   b. 如果指定了 xmlattributes-function，那么会对其求值，结果是属性节点序列。
   c. 将对每个 *element-content-expression* 求值，结果将转换为如下所示的节点序列:
      - 如果结果类型并非 XML，那么它将转换为 XML 文本节点，其内容是根据将 SQL 数据值映射至 XML 数据值（请参阅“数据类型之间的强制转换”中描述从非 XML 值强制转换为 XML 值的表）中的规则映射至 XML 的 *element-content-expression* 的结果。
      - 如果结果类型为 XML，那么一般来说结果是项序列。该序列中的某些项可能是文档节点。序列中的每个文档节点将替换为其顶级子代序列。对于结果序列中的每个节点，将为该节点构造新的深层副本，包括其子代和属性。每个被复制节点具有新的节点标识。被复制元素和属性节点将保留其类型注释。对于由序列中返回的一个或多个原子值组成的每个相邻序列，将构造新的文本节点（包含将每个原子值强制转换为字符串的结果），并会在相邻值之间插入单个空白字符。内容序列中的相邻文本节点将通过并置内容（相邻值之间不插入空格）来合并成单个文本节点。并置后，将从内容序列中删除内容为零长度字符串的任何文本节点。
   d. XML 属性的结果序列和所有 *element-content-expression* 指定结果序列将并置成为一个序列，称为内容序列。内容序列中的相邻文本节点的任何序列将合并成单个文本节点。如果所有 *element-content-expression* 自变量都是空字符串，或者未指定 *element-content-expression* 自变量，那么会返回空元素。
   e. 内容序列只能包含后跟属性节点的属性节点（SQLSTATE 10507）。内容序列中出现的属性节点将成为新元素节点的属性。在这些属性节点中，一定不能有两个或两个以上属性节点同名（SQLSTATE 10503）。如果名称空间 URI 不在已构造元素的名称空间作用域中，那么将创建对应于属性节点名称中使用的任何名称空间的名称空间声明。
   f. 内容序列中的元素、文本、注释和处理指令将成为已构造元素节点的子代。
   g. 已构造元素节点的类型注释被指定为 `xs:anyType`，它的每个属性的类型注释被指定为 `xdt:untypedAtomic`。已构造元素节点的节点名是在 NAME 关键字之后指定的元素名称。
3. **在 XMLELEMENT 中使用名称空间的规则**: 考虑下列有关名称空间作用域限定的规则:

- XMLNAMESPACES 声明中声明的名称空间是由 XMLELEMENT 函数构造的元素节点的名称空间作用域。如果元素节点已序列化，那么它的每个名称空间作用域将作为名称空间属性序列化，除非它是元素节点父代的名称空间作用域，而父代元素也已序列化。
- 如果 XMLQUERY 或 XMLEXISTS 在 *element-content-expression* 中，那么这些名称空间将成为 XMLQUERY 或 XMLEXISTS 的 XQuery 表达式的静态已知名称空间。静态已知名称空间用于解析 XQuery 表达式中的 QNames。如果 XQuery 序言在 XQuery 表达式作用域中使用同一前缀声明名称空间，那么序言中声明的名称空间将覆盖 XMLNAMESPACES 声明中声明的名称空间。
- 如果已构造元素的属性来自 *element-content-expression*，那么其名称空间可能尚未声明为已构造元素的名称空间作用域，在此情况下，将其为其创建新的名称空间。如果这会导致冲突（意味着属性名称的前缀已被名称空间作用域绑定至另一 URI），那么 DB2 将生成不会导致这类冲突的前缀，而属性名称中使用的前缀将更改为新前缀，并将为此新前缀创建名称空间。生成的新前缀将遵循以下模式："db2ns-xx"，其中"x"是从集合 [A-Z、a-z 和 0-9] 中选择的字符。例如:

  ```
  VALUES XMLELEMENT(
    NAME "c", XMLQUERY(
      'declare namespace ipo="www.ipo.com"; $m/ipo:a/@ipo:b'
        PASSING XMLPARSE(
          DOCUMENT '<tst:a xmlns:tst="www.ipo.com" tst:b="2"/>'
        ) AS "m"
    )
  )
  ```

  返回:

  ```
  <c xmlns:tst="www.ipo.com" tst:b="2"/>
  ```

  第二个示例:

  ```
  VALUES XMLELEMENT(
    NAME "tst:c", XMLNAMESPACES(
      'www.tst.com' AS "tst"
    ),
    XMLQUERY(
      'declare namespace ipo="www.ipo.com"; $m/ipo:a/@ipo:b'
        PASSING XMLPARSE(
          DOCUMENT '<tst:a xmlns:tst="www.ipo.com" tst:b="2"/>'
        ) AS "m"
    )
  )
  ```

  返回:

  ```
  <tst:c xmlns:tst="www.tst.com" xmlns:db2ns-a1="www.ipo.com"
    db2ns-a1:b="2"/>
  ```

示例:

注: XMLELEMENT 不会在输出中插入空格或换行符。所有示例输出都将格式化以增强可读性。

- 使用 NULL ON NULL 选项来构造元素。

  ```
  SELECT E.FIRSTNME, E.LASTNAME, XMLELEMENT(
    NAME "Emp", XMLELEMENT(
      NAME "firstname", E.FIRSTNME
    ),
    XMLELEMENT(
      NAME "lastname", E.LASTNAME
  ```

```
    )
    OPTION NULL ON NULL
  )
  AS "Result"
  FROM EMPLOYEE E
  WHERE E.EDLEVEL = 12
```

此查询将生成以下结果:

```
FIRSTNME     LASTNAME        Emp
JOHN         PARKER          <Emp><firstname>JOHN</firstname>
                               <lastname>PARKER</lastname></Emp>
MAUDE        SETRIGHT        <Emp><firstname>MAUDE</firstname>
                               <lastname>SETRIGHT</lastname></Emp>
MICHELLE     SPRINGER        <Emp><firstname>MICHELLE</firstname>
                               <lastname>SPRINGER</lastname></Emp>
```

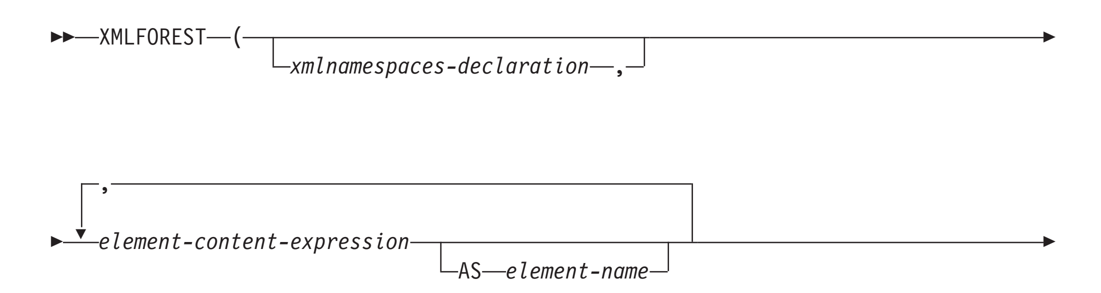
- 生成带有作为子元素嵌套的元素列表的元素。

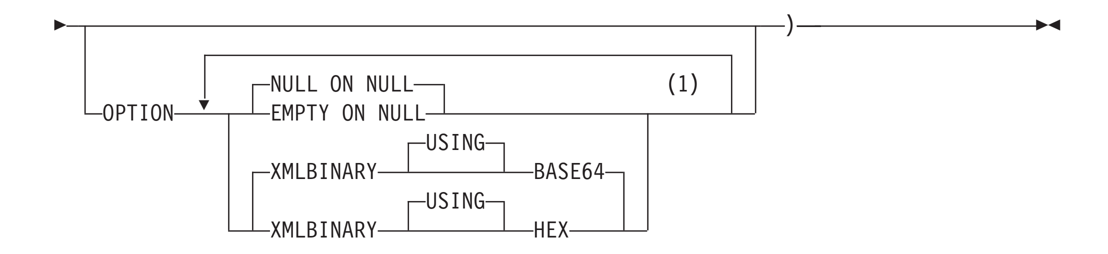
```
SELECT XMLELEMENT(
  NAME "Department", XMLATTRIBUTES(
    E.WORKDEPT AS "name"
  ),
  XMLAGG(
    XMLELEMENT(
      NAME "emp", E.FIRSTNME
    )
    ORDER BY E.FIRSTNME
  )
)
AS "dept_list"
FROM EMPLOYEE E
WHERE E.WORKDEPT IN ('A00', 'B01')
GROUP BY WORKDEPT
```

此查询将生成以下结果:

```
dept_list
<Department name="A00">
<emp>CHRISTINE</emp>
<emp>SEAN</emp>
<emp>VINCENZO</emp>
</Department>
<Department name="B01">
<emp>MICHAEL</emp>
</Department>
```

XMLFOREST





注:

1　不能多次指定同一子句。

模式为 SYSIBM。不能将函数名指定为限定名。

XMLFOREST 函数将返回 XML 值，即 XQuery 元素节点序列。

xmlnamespaces-declaration
: 指定 XMLNAMESPACES 声明生成的 XML 名称空间声明。声明的名称空间在 XMLFOREST 函数作用域中。不管是否出现在另一子查询中，这些名称空间都适用于 XMLFOREST 函数内的任何嵌套 XML 函数。

  如果未指定 *xmlnamespaces-declaration*，那么名称空间声明与已构造元素无关。

element-content-expression
: 生成的 XML 元素节点的内容是由表达式指定的。*element-content-expression* 的数据类型不能是结构化类型（SQLSTATE 42884）。该表达式可以是任何 SQL 表达式。如果该表达式不是简单列引用，那么必须指定元素名称。

AS *element-name*
: 指定 XML 元素名称作为 SQL 标识。元素名称必须使用 XML 限定名称或 QName 格式（SQLSTATE 42634）。有关有效名称的更多详细信息，请参阅 W3C XML 名称空间规范。如果是限定名称，那么必须在作用域中声明名称空间前缀（SQLSTATE 42635）。如果未指定 *element-name*，那么 *element-content-expression* 必须是列名（SQLSTATE 42703）。元素名称是根据列名使用从列名至 QName 的完全转义映射创建的。

OPTION
: 指定用于构造 XML 元素的其他选项。如果未指定 OPTION 子句，那么缺省值为 NULL ON NULL XMLBINARY USING BASE64。此子句对 *element-content-expression* 中指定的嵌套 XMLELEMENT 调用没有影响。

  EMPTY ON NULL 或 NULL ON NULL
  : 如果每个 *element-content-expression* 的值都是空值，那么指定是返回空值还是空元素。此选项仅影响元素内容的空值处理，而不会影响属性值。缺省值为 NULL ON NULL。

    EMPTY ON NULL
    : 如果每个 *element-content-expression* 的值都为空，那么会返回空元素。

    NULL ON NULL
    : 如果每个 *element-content-expression* 的值为空，那么会返回空值。

XMLBINARY USING BASE64 或 XMLBINARY USING HEX

指定二进制输入数据的采用编码、带有 FOR BIT DATA 属性的字符串或基于其中一种类型的单值类型。该编码适用于元素内容或属性值。缺省值为 XMLBINARY USING BASE64。

XMLBINARY USING BASE64

根据对 XML 模式类型 xs:base64Binary 编码的定义，指定采用编码为基本 64 位字符。基本 64 位代码使用 US-ASCII 的由 65 个字符组成的子集（10 个数字、26 个小写字符、26个大写字符以及"+"和" "）来表示每个二进制数据或位数据的 6 位以及子集中的一个可打印字符。这些字符已被选中，所以它们是以可通用的方式表示的。使用此方法时，编码数据的大小比原始二进制数据或位数据大 33%。

XMLBINARY USING HEX

根据对 XML 模式类型 xs:hexBinary 编码的定义，指定采用编码为十六进制字符。十六进制编码使用两个十六进制字符来表示每个字节（8 位）。使用此方法时，编码数据的大小是原始二进制数据或位数据的两倍。

此函数采用名称空间声明、构成该名称的一个或多个自变量以及一个或多个元素节点的元素内容的可选集合。结果是包含 XQuery 元素或空值的序列的 XML 序列。

结果的数据类型为 XML。如果任何 *element-content-expression* 自变量可为空，那么结果可为空；如果所有 *element-content-expression* 自变量值为空并且 NULL ON NULL 选项生效，那么结果为空。

XMLFOREST 函数可使用 XMLCONCAT 和 XMLELEMENT 来表示。例如，下列两个表达式在语义上是等价的。

```
XMLFOREST(xmlnamespaces-declaration, arg1 AS name1, arg2 AS name2 ...)
```

```
XMLCONCAT(
  XMLELEMENT(
    NAME name1, xmlnamespaces-declaration, arg1
  ),
  XMLELEMENT(
    NAME name2, xmlnamespaces-declaration, arg2
  )
  ...
)
```

注:

1. 构造将作为另一元素（用于定义缺省名称空间）的内容进行复制的元素时，应在被复制元素中显式取消声明缺省名称空间，以避免因为从新的父代元素继承缺省名称空间而可能导致的错误。预定义名称空间前缀（"xs"、"xsi"、"xml"和"sqlxml"）在使用时也必须显式声明。

示例:

注: XMLFOREST 不会在输出中插入空格或换行符。所有示例输出都将格式化以增强可读性。

- 使用缺省名称空间构造元素群。

```
SELECT EMPNO,
  XMLFOREST(
    XMLNAMESPACES(
      DEFAULT 'http://hr.org', 'http://fed.gov' AS "d"
    ),
```

```
      LASTNAME, JOB AS "d:job"
    )
  AS "Result"
  FROM EMPLOYEE
  WHERE EDLEVEL = 12
```

此查询将生成以下结果:

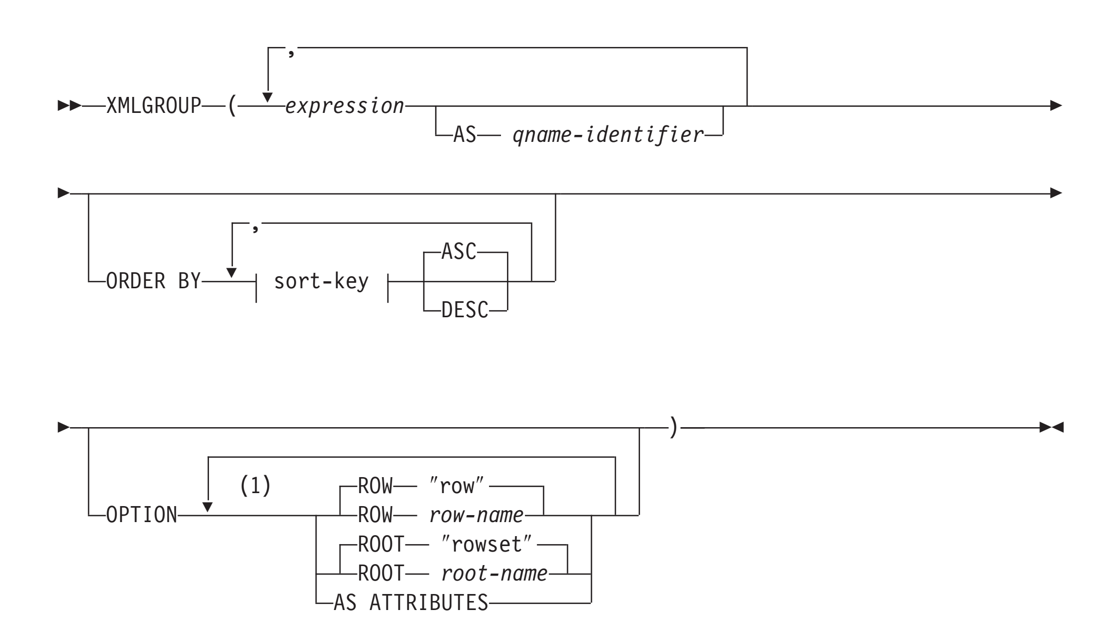
```
EMPNO  Result
000290 <LASTNAME xmlns="http://hr.org" xmlns:d="http://fed.gov">PARKER
  </LASTNAME>
<d:job xmlns="http://hr.org" xmlns:d="http://fed.gov">OPERATOR</d:job>

000310 <LASTNAME xmlns="http://hr.org" xmlns:d="http://fed.gov">SETRIGHT
  </LASTNAME>
<d:job xmlns="http://hr.org" xmlns:d="http://fed.gov">OPERATOR</d:job>

200310 <LASTNAME xmlns="http://hr.org" xmlns:d="http://fed.gov">SPRINGER
  </LASTNAME>
<d:job xmlns="http://hr.org" xmlns:d="http://fed.gov">OPERATOR</d:job>
```

XMLGROUP

XMLGROUP 函数返回带有单个 XQuery 文档节点的 XML 值，该文档节点包含一个顶级元素节点。这是一个聚集表达式，将从每行映射至行子元素的一组行中返回单根 XML 文档。



注:

1 不能多次指定同一子句。

模式为 SYSIBM。不能将函数名指定为限定名。

expression

每个已生成 XML 元素节点的内容（或每个已生成属性的值）由表达式指定。*expression* 的数据类型不能是结构化类型（SQLSTATE 42884）。该表达式可以是任何 SQL 表达式。如果该表达式不是简单列引用，那么必须指定 *qname-identifier*。

AS *qname-identifier*

指定 XML 元素名称或属性名称作为 SQL 标识。*qname-identifier* 必须使用 XML 限定名称或 QName 格式（SQLSTATE 42634）。请参阅 W3C XML 名称空间规范以了解有关有效名称的更多详细信息。如果是限定名称，那么必须在作用域中声明名称空间前缀（SQLSTATE 42635）。如果未指定 *qname-identifier*，那么 *expression* 必须是列名（SQLSTATE 42703）。元素名称或属性名称是根据列名使用从列名至 QName 的完全转义映射创建的。

OPTION

指定用于构造 XML 值的其他选项。如果未指定 OPTION 子句，那么缺省行为适用。

ROW *row-name*

指定每行将映射至的元素的名称。如果未指定此选项，那么缺省元素名称为"row"。

ROOT *root-name*

指定根元素节点的名称。如果未指定此选项，那么缺省根元素名称为"rowset"。

AS ATTRIBUTES

指定每个表达式将映射至列名或 *qname-identifier* 充当属性名的属性值。

ORDER BY

指定聚集中处理的同一分组集合中的行顺序。如果省略了 ORDER BY 子句，或者 ORDER BY 子句不能区分列数据的顺序，那么同一分组集合中的行将任意排序。

sort-key

排序键可以是列名或 *sort-key-expression*。注意，如果排序键为常量，那么它不会像普通 ORDER BY 子句中那样引用输出列的位置，但它只是一个常量，这意味着没有排序键。

注意

缺省行为将定义结果集与 XML 值之间的简单映射。有关函数行为的一些附加注释适用:

- 缺省情况下，每行将变换为名为"row"的 XML 元素，而每列将变换为列名充当元素名称的嵌套元素。
- 空值处理行为是 NULL ON NULL。列中的空值将映射至缺少子元素的位置。如果所有列值都为空，那么不会生成行元素。
- BLOB 和 FOR BIT DATA 数据类型的二进制编码方案为 base64Binary 编码。
- 缺省情况下，对应于组中的各行的元素是名为"rowset"的根元素的子代。
- 根元素中的行子元素顺序与查询结果集中返回的行顺序相同。
- 文档节点将以隐式方式添加至根元素，以使 XML 结果成为格式良好的单根 XML 文档。

示例

假定下表 T1 的整数列 C1 和 C2 包含存储为关系格式的数字数据:

```
C1          C2
----------- -----------
          1           2
          -           2
```

```
       1           -
       -           -

  4 record(s) selected.
```

- 以下示例显示具有缺省行为的 XMLGroup 查询和输出片段并使用单个顶级元素来表示该表。:

```
SELECT XMLGROUP(C1, C2)FROM T1
```

```
<rowset>
   <row>
      <C1>1</C1>
      <C2>2</C2>
   </row>
   <row>
      <C2>2</C2>
   </row>
   <row>
      <C1>1</C1>
   </row>
</rowset>

  1 record(s) selected.
```

- 以下示例显示具有属性中心映射的 XMLGroup 查询和输出片段。关系数据将映射至元素属性，而不是像先前示例中一样作为嵌套元素出现:

```
SELECT XMLGROUP(C1, C2 OPTION AS ATTRIBUTES) FROM T1
```

```
<rowset>
   <row C1="1" C2="2"/>
   <row C2="2"/>
   <row C1="1"/>
</rowset>

  1 record(s) selected.
```

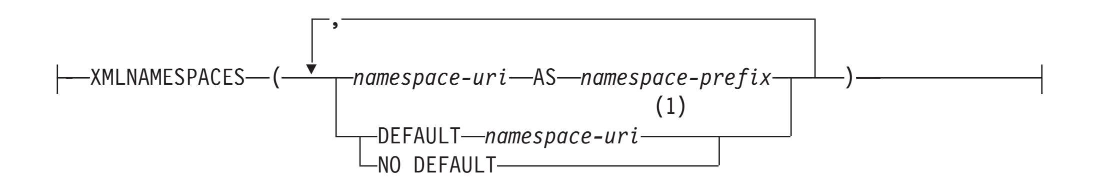
- 以下示例显示缺省 <rowset> 根元素替换为 <document> 而 <row> 元素替换为 <entry> 的 XMLGroup 和输出片段。列 C1 和 C2 作为 <column1> 和 <column2> 元素返回，而返回集按列 C1 排序:

```
SELECT XMLGROUP(
   C1 AS "column1", C2 AS "column2"
   ORDER BY C1 OPTION ROW "entry" ROOT "document")
        FROM T1
```

```
<document>
   <entry>
      <column1>1</column1>
      <column2>2</column2>
   </entry>
   <entry>
      <column1>1</column1>
   </entry>
   <entry>
      <column2>2</column2>
    </entry>
</document>
```

XMLNAMESPACES

xmlnamespaces-declaration:



注:

1　DEFAULT 或 NO DEFAULT 只能在 XMLNAMESPACES 的自变量中指定一次。

模式为 SYSIBM。不能将声明名称指定为限定名称。

XMLNAMESPACES 声明通过自变量构造名称空间声明。此声明只能用作特定函数（XMLELEMENT、XMLFOREST 和 XMLTABLE）的自变量。结果是包含每个非空输入值的名称空间作用域的一个或多个 XML 名称空间声明。

namespace-uri
: 指定名称空间通用资源标识（URI）作为 SQL 字符串常量。如果此字符串常量用于 *namespace-prefix*，那么一定不能为空（SQLSTATE 42815）。

namespace-prefix
: 指定名称空间前缀。前缀是必须为 XML NCName 格式的 SQL 标识（SQLSTATE 42634）。有关有效名称的更多详细信息，请参阅 W3C XML 名称空间规范。前缀不能为 `xml` 或 `xmlns`，并且前缀在名称空间声明列表中必须唯一（SQLSTATE 42635）。

DEFAULT *namespace-uri*
: 指定要在此名称空间声明的作用域中使用的缺省名称空间。除非在嵌套作用域中被另一 DEFAULT 声明或 NO DEFAULT 声明覆盖，否则 *namespace-uri* 将应用于该作用域中的未限定名称。

NO DEFAULT
: 指定不在此名称空间声明的作用域中使用缺省名称空间。除非在嵌套作用域中被 DEFAULT 声明覆盖，否则该作用域中没有缺省名称空间。

结果的数据类型为 XML。结果是每个指定名称空间的 XML 名称空间声明。结果不能为空。

示例:

注: XMLNAMESPACES 不会在输出中插入空格或换行符。所有示例输出都将格式化以增强可读性。

- 将生成 XML 元素 adm:employee 和 XML 属性 adm:department，这两项都与前缀为 adm 的名称空间相关联。

```
SELECT EMPNO, XMLELEMENT(
  NAME "adm:employee", XMLNAMESPACES(
    'http://www.adm.com' AS "adm"
  ),
  XMLATTRIBUTES(
    WORKDEPT AS "adm:department"
  ),
LASTNAME
)
FROM EMPLOYEE
WHERE JOB = 'ANALYST'
```

此查询将生成以下结果:

```
000130 <adm:employee xmlns:adm="http://www.adm.com" adm:department="C01">
  QUINTANA</adm:employee>
000140 <adm:employee xmlns:adm="http://www.adm.com" adm:department="C01">
  NICHOLLS</adm:employee>
200140 <adm:employee xmlns:adm="http://www.adm.com" adm:department="C01">
  NATZ</adm:employee>
```

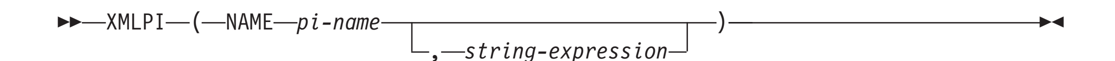
- 将生成 XML 元素“employee”和子元素“job”，XML 元素“employee”与缺省名称空间相关联，子元素“job”不使用缺省名称空间，但其子元素“department”使用缺省名称空间。

```
SELECT EMP.EMPNO, XMLELEMENT(
  NAME "employee", XMLNAMESPACES(
    DEFAULT 'http://hr.org'
  ),
  EMP.LASTNAME, XMLELEMENT(
    NAME "job", XMLNAMESPACES(
      NO DEFAULT
    ),
    EMP.JOB, XMLELEMENT(
      NAME "department", XMLNAMESPACES(
        DEFAULT 'http://adm.org'
      ),
      EMP.WORKDEPT
    )
  )
)
FROM EMPLOYEE EMP
WHERE EMP.EDLEVEL = 12
```

此查询将生成以下结果:

```
000290 <employee xmlns="http://hr.org">PARKER<job xmlns="">OPERATOR
  <department xmlns="http://adm.org">E11</department></job></employee>
000310 <employee xmlns="http://hr.org">SETRIGHT<job xmlns="">OPERATOR
  <department xmlns="http://adm.org">E11</department></job></employee>
200310 <employee xmlns="http://hr.org">SPRINGER<job xmlns="">OPERATOR
  <department xmlns="http://adm.org">E11</department></job></employee>
```

XMLPI



模式为 SYSIBM。不能将函数名指定为限定名。

XMLPI 函数将返回带有单个 XQuery 处理指令节点的 XML 值。

NAME *pi-name*

指定处理指令的名称。该名称是必须为 XML NCName 格式的 SQL 标识（SQLSTATE 42634）。有关有效名称的更多详细信息，请参阅 W3C XML 名称空间规范。名称在任何案例组合中都不能为单词“xml”（SQLSTATE 42634）。

string-expression

返回值为字符串的表达式。生成的字符串将转换为 UTF-8，并且必须符合 XML 1.0 规则中指定 XML 处理指令的内容（SQLSTATE 2200T）：

- 该字符串一定不能包含“?>”，原因是此子串将终止处理指令。

- 字符串的每个字符可以是超大字符集块 X’FFFE’ 和 X’FFFF’ 以外的任何 Unicode 字符。

  生成的字符串将成为已构造处理指令节点的内容。

结果的数据类型为 XML。如果 *string-expression* 的结果可为空，那么结果可为空；如果 *string-expression* 的结果为空，那么结果为空。如果 *string-expression* 是空字符串或者未指定，那么会返回空处理指令。

示例:

- 生成 XML 处理指令节点。

```
SELECT XMLPI(
  NAME "Instruction", 'Push the red button'
)
FROM SYSIBM.SYSDUMMY1
```

  此查询将生成以下结果:

```
<?Instruction Push the red button?>
```

- 生成空 XML 指令节点。

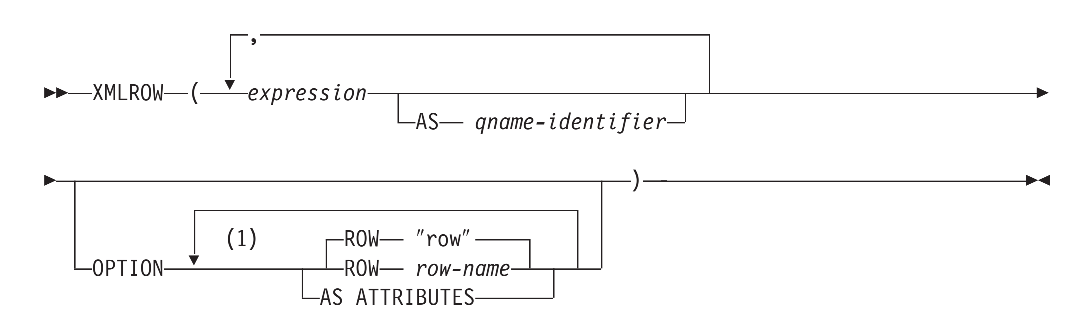
```
SELECT XMLPI(
  NAME "Warning"
)
FROM SYSIBM.SYSDUMMY1
```

  此查询将生成以下结果:

```
<?Warning ?>
```

XMLROW

XMLROW 函数返回带有单个 XQuery 文档节点的 XML 值，该文档节点包含一个顶级元素节点。



注:

1　　不能多次指定同一子句。

模式为 SYSIBM。不能将函数名指定为限定名。

expression
: 每个已生成 XML 元素节点的内容由表达式指定。表达式的数据类型不能是结构化类型（SQLSTATE 42884）。该表达式可以是任何 SQL 表达式。如果该表达式不是简单列引用，那么必须指定元素名称。

AS *qname-identifier*

指定 XML 元素名称或属性名称作为 SQL 标识。*qname-identifier* 必须使用 XML 限定名称或 QName 格式（SQLSTATE 42634）。请参阅 W3C XML 名称空间规范以了解有关有效名称的更多详细信息。如果是限定名称，那么必须在作用域中声明名称空间前缀（SQLSTATE 42635）。如果未指定 *qname-identifier*，那么 *expression* 必须是列名（SQLSTATE 42703）。元素名称或属性名称是根据列名使用从列名至 QName 的完全转义映射创建的。

OPTION

指定用于构造 XML 值的其他选项。如果未指定 OPTION 子句，那么缺省行为适用。

AS ATTRIBUTES

指定每个表达式将映射至列名或 *qname-identifier* 充当属性名的属性值。

ROW *row-name*

指定每行将映射至的元素的名称。如果未指定此选项，那么缺省元素名称为“row”。

注意

缺省情况下，结果集中的每行映射至 XML 值，如下所示:

- 每行将变换为名为“row”的 XML 元素，而每列将变换为列名充当元素名称的嵌套元素。
- 空值处理行为是 NULL ON NULL。列中的空值将映射至缺少子元素的位置。如果所有列值都为空，那么函数将返回空值。
- BLOB 和 FOR BIT DATA 数据类型的二进制编码方案为 base64Binary 编码。
- 文档节点将以隐式方式添加至行元素，以使 XML 结果成为格式良好的单根 XML 文档。

示例

假定下表 T1 的列 C1 和 C2 包含存储为关系格式的数字数据:

```
C1          C2
----------- -----------
          1           2
          -           2
          1           -
          -           -

  4 record(s) selected.
```

- 以下示例显示具有缺省行为的 XMLRow 查询和输出片段并使用行元素序列来表示该表。:

```
SELECT XMLROW(C1, C2) FROM T1

<row><C1>1</C1><C2>2</C2></row>
<row><C2>2</C2></row>
<row><C1>1</C1></row>

  4 record(s) selected.
```

- 以下示例显示具有属性中心映射的 XMLRow 查询和输出片段。关系数据将映射至元素属性，而不是像先前示例中一样作为嵌套元素出现:

```
SELECT XMLROW(C1, C2 OPTION AS ATTRIBUTES) FROM T1
```

```
<row C1="1" C2="2"/>
<row C2="2"/>
<row C1="1"/>

  4 record(s) selected.
```

- 以下示例显示缺省 <row> 元素替换为 <entry> 的 XMLRow 和输出片段。列 C1 和 C2 将作为 <column1> 和 <column2> 元素返回，而 C1 和 C2 的总和将在 <total> 元素中返回：

```
SELECT XMLROW(
   C1 AS "column1", C2 AS "column2",
   C1+C2 AS "total" OPTION ROW "entry")
FROM T1
```

```
<entry><column1>1</column1><column2>2</column2><total>3</total></entry>
<entry><column2>2</column2></entry>
<entry><column1>1</column1></entry>

  4 record(s) selected.
```

XMLTEXT

```
►►—XMLTEXT—(—string-expression—)——————————————————————►◄
```

模式为 SYSIBM。不能将函数名指定为限定名。

XMLTEXT 函数将返回具有单个 XQuery 文本节点的 XML 值，该文本节点将输入自变量作为内容。

string-expression
  其值为字符串类型 CHAR、VARCHAR 或 CLOB 的表达式。

结果的数据类型为 XML。如果 *string-expression* 的结果可为空，那么结果可为空；如果输入值为空，那么结果为空。如果 *string-expression* 的结果为空字符串，那么结果值为空文本节点。

示例：

- 创建简单 XMLTEXT 查询。

```
VALUES(
  XMLTEXT(
    'The stock symbol for Johnson&Johnson is JNJ.'
  )
)
```

此查询将生成以下序列化结果：

```
1
---------------------------------------------------
The stock symbol for Johnson&amp;Johnson is JNJ.
```

注意，文本节点经过序列化后，“&”符号将映射至“&”。

- 将 XMLTEXT 与 XMLAGG 配合使用以构造混合内容。假定表 T 的内容如下所示：

```
seqno  plaintext                                                  emphtext
------ ---------------------------------------------------------- -------------
1      This query shows how to construct                          mixed content
2       using XMLAGG and XMLTEXT. Without                         XMLTEXT
3      XMLAGG will not have text nodes to group with other nodes,  mixed content
       therefore, cannot generate
```

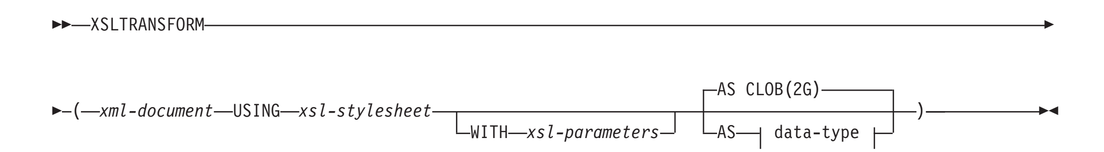
```
SELECT XMLELEMENT(
  NAME "para", XMLAGG(
    XMLCONCAT(
      XMLTEXT(
        PLAINTEXT
      ),
      XMLELEMENT(
        NAME "emphasis", EMPHTEXT
      )
    )
    ORDER BY SEQNO
  ), '.'
) AS "result"
FROM T
```

此查询将生成以下结果:

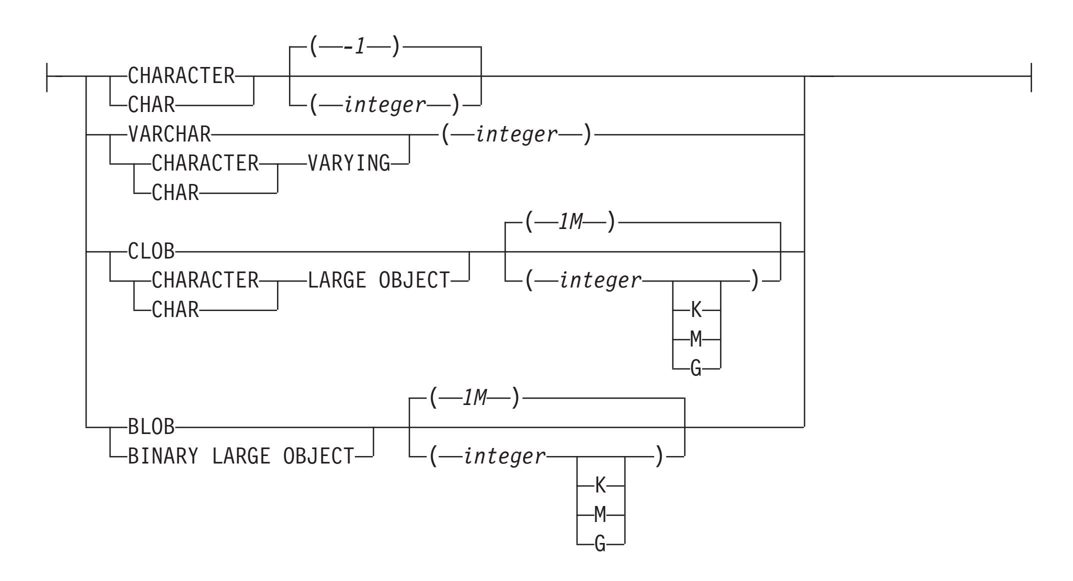
```
result
-------------------------------------------------------------------------------
<para>This query shows how to construct <emphasis>mixed content</emphasis>
using XMLAGG and XMLTEXT. Without <emphasis>XMLTEXT</emphasis> , XMLAGG
will not have text nodes to group with other nodes, therefore, cannot generate
<emphasis>mixed content</emphasis>.</para>
```

XSLTRANSFORM

使用 XSLTRANSFORM 将 XML 数据转换为其他格式，包括将符合一种 XML 模式的 XML 文档转换为符合另一种模式的文档。



data-type:



模式为 SYSIBM。不能将此函数指定为限定名。

XSLTRANSFORM 函数将 XML 文档变换成另一种数据格式。可以将数据变换成 XSLT 处理器可处理的任何格式，包括但不限于 XML、HTML 或纯文本。

XSLTRANSFORM 使用的所有路径对于数据库系统来说都是内部路径。目前，不能直接对外部文件系统中的文件或样式表使用此命令。

xml-document

一个表达式，它返回数据类型为 XML、CHAR、VARCHAR、CLOB 或 BLOB 且结构良好的 XML 文档。这是使用 *xsl-stylesheet* 中指定的 XSL 样式表进行变换的文档。

注:

XML 文档必须至少是结构良好的单根文档。

xsl-stylesheet

一个表达式，它返回数据类型为 XML、CHAR、VARCHAR、CLOB 或 BLOB 且结构良好的 XML 文档。该文档是符合 W3C XSLT V1.0 建议的 XSL 样式表。不支持包含 XQUERY 语句或 `xsl:include` 声明的样式表。此样式表用于变换 *xml-document* 中指定的值。

xsl-parameters

一个表达式，它返回数据类型为 XML、CHAR、VARCHAR、CLOB 或 BLOB 且结构良好的 XML 文档或空值。这是为 *xsl-stylesheet* 中指定的 XSL 样式表提供参数值的文档。可以将参数值指定为属性或文本节点。

参数文档的语法如下所示:

```
<params xmlns="http://www.ibm.com/XSLTransformParameters">
<param name="..." value="..."/>
<param name="...">enter value here</param>
  ...
</params>
```

注:

样式表文档中必须包含 xsl:param 元素，并且具有与参数文档中所指定的名称属性值相匹配的名称属性值。

AS *data-type*
指定结果数据类型。指定的结果数据类型的隐式或显式长度属性必须足够长，能够包含变换后的输出（SQLSTATE 22001）。缺省结果数据类型为 CLOB（2G）。

注:

如果 *xml-document* 参数或 *xsl-stylesheet* 参数为空，那么结果将为空。

将上述任何文档存储在 CHAR、VARCHAR 或 CLOB 列中时可能会导致代码页转换，从而可能导致字符丢失。

示例

此示例演示如何将 XSLT 用作格式化引擎。要进行设置，先将下面两个示例文档插入到数据库中。

```
INSERT INTO XML_TAB VALUES
(1,
          '<?xml version="1.0"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation = "/home/steffen/xsd/xslt.xsd">
<student studentID="1" firstName="Steffen" lastName="Siegmund"
    age=â€23â€ university=â€Rostockâ€/>
</students>',
    '<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="showUniversity"/>
<xsl:template match="students">
                 <html>
                    <head/>
                       <body>
                                <h1><xsl:value-of select="$headline"/></h1>
                              <table border="1">
                                      <th>
                                          <tr>
                                <td width="80">StudentID</td>
                                <td width="200">First Name</td>
                                <td width="200">Last Name</td>
                                <td width="50">Age</td>
                                <xsl:choose>
       <xsl:when test="$showUniversity ='true'">
                                               <td width="200">University</td>
                                    </xsl:when>
       </xsl:choose>
                             </tr>
                                </th>
                                <xsl:apply-templates/>
                              </table>
                       </body>
                </html>
               </xsl:template>
                      <xsl:template match="student">
                    <tr>
                  <td><xsl:value-of select="@studentID"/></td>
                  <td><xsl:value-of select="@firstName"/></td>
                  <td><xsl:value-of select="@lastName"/></td>
```

```
                  <td><xsl:value-of select="@age"/></td>
                  <xsl:choose>
                      <xsl:when test="$showUniversity = 'true' ">
                        <td><xsl:value-of select="@university"/></td>
                      </xsl:when>
                  </xsl:choose>
                         </tr>
                    </xsl:template>
</xsl:stylesheet>'
);
```

下面调用 XSLTRANSFORM 函数以将 XML 数据转换为 HTML 并显示出来。

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC AS CLOB(1M)) FROM XML_TAB;
```

结果为以下文档:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">First Name</td>
<td width="200">Last Name</td>
<td width="50">Age</td>
</tr>
</th>
 <tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>23</td>
</tr>
              </table>
</body>
</html>
```

在此示例中，输出为 HTML 并且这些参数仅影响产生的 HTML 内容及提供的数据。这样它会演示如何将 XSLT 用作最终用户输出的格式化引擎。

使用说明:

此函数**不**会在高性能的应用程序中使用，并且**不能**替换该应用程序服务器中相似的功能。

附录 C. XSR 存储过程和命令

下列各节描述 DB2 XSR 存储过程和命令的语法。

有关使用存储过程和命令的信息，请参阅第 171 页的『XSR 对象』。

XSR 存储过程

XSR_REGISTER

```
►►—XSR_REGISTER—(—rschema—,—name—,—schemalocation—,—content—,——————►

►–docproperty—)——————————————————————————————————————————————————►◄
```

模式为 SYSPROC。

XSR_REGISTER 过程是在 XML 模式注册过程中要调用的第一个存储过程，它将向 XML 模式存储库（XSR）注册 XML 模式。

权限

该过程的调用者的授权标识必须至少具有下列其中一种权限或特权:

- DBADM 权限。
- IMPLICIT_SCHEMA 数据库权限（如果 SQL 模式不存在）。
- CREATEIN 特权（如果 SQL 模式存在）。

rschema
: VARCHAR (128) 类型的输入和输出参数，它指定 XML 模式的 SQL 模式。SQL 模式是用来在 XSR 中标识此 XML 模式的 SQL 标识的一部分。（SQL 标识的另一部分是由 name 参数提供的。）此参数的值可以为空，这表示使用了缺省 SQL 模式（它是在专用寄存器 CURRENT SCHEMA 中定义的）。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。不能将以字符串“SYS”开头的关系模式用于此值。因为 XSR 对象与 XML 模式存储库外部的对象位于不同的名称空间中，所以 XSR 对象不会与存在于 XSR 外部的数据库对象之间发生名称冲突。

name
: VARCHAR(128) 类型的输入和输出参数，它指定 XML 模式的名称。XML 模式的完整 SQL 标识为 *rschema.name*，并且对于 XSR 中的所有对象应该是唯一的。此参数接受空值。当为此参数提供空值时，就会生成一个唯一值并存储在 XSR 中。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。

schemalocation
: VARCHAR (1000) 类型的输入参数，用于指示 XML 模式主文档的模式位置，它可以具有空值。此参数是 XML 模式的外部名，即，可以在 XML 实例文档中使用 xsi:schemaLocation 属性来标识主文档。

content

BLOB (30M) 类型的输入参数，包含 XML 模式主文档的内容。此参数的值不能为空；必须提供一个 XML 模式文档。

docproperty

BLOB (5M) 类型的输入参数，指示 XML 模式主文档的属性。此参数可以具有空值；否则，值为 XML 文档。

示例：以下示例说明如何从命令行调用 XSR_REGISTER 过程:

```
CALL SYSPROC.XSR_REGISTER(
       'user1',
       'POschema',
       'http://myPOschema/PO.xsd',
       :content_host_var,
       :docproperty_host_var)
```

示例：以下示例说明如何从 Java 应用程序调用 XSR_REGISTER 过程:

```
stmt = con.prepareCall("CALL SYSPROC.XSR_REGISTER (?, ?, ?, ?, ?)");
String xsrObjectName = "myschema1";
String xmlSchemaLocation = "po.xsd";
stmt.setNull(1, java.sql.Types.VARCHAR);
stmt.setString(2, xsrObjectName);
stmt.setString(3, xmlSchemaLocation);
stmt.setBinaryStream(4, buffer, (int)length);
stmt.setNull(5, java.sql.Types.BLOB);
stmt.registerOutParameter(1, java.sql.Types.VARCHAR);
stmt.registerOutParameter(2, java.sql.Types.VARCHAR);
stmt.execute();
```

XSR_ADDSCHEMADOC

```
►►—XSR_ADDSCHEMADOC—(—rschema—,—name—,—schemalocation—,—content—,———►

►—docproperty—)——————————————————————————————————————————————————————►◄
```

模式为 SYSPROC。

XML 模式存储库（XSR）中的每个 XML 模式都可以由一个或多个 XML 模式文档组成。在 XML 模式由多个文档组成的情况下，XSR_ADDSCHEMADOC 存储过程用来添加每个 XML 模式而不是 XML 模式主文档。

权限

该过程的调用者的授权标识必须是 SYSCAT.XSROBJECTS 目录视图中记录的 XSR 对象的所有者。

rschema

VARCHAR (128) 类型的输入参数，它指定 XML 模式的 SQL 模式。SQL 模式是用来在 XSR 中标识此 XML 模式的 SQL 标识的一部分，它将变成完整状态。（SQL 标识的另一部分是由 name 参数提供的。）此参数的值可以为空，这表示使用了缺省 SQL 模式（它是在专用寄存器 CURRENT SCHEMA 中定义的）。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。因为 XSR 对象与 XML 模式存储库外部的对象位于不同的名称空间中，所以 XSR 对象不会与存在于 XSR 外部的数据库对象之间发生名称冲突。

name
: VARCHAR(128) 类型的输入参数，它指定 XML 模式的名称。XML 模式的完整 SQL 标识为 *rschema.name*。必须通过调用 XSR_REGISTER 存储过程已经获得了 XML 模式名称，并且 XML 模式注册尚不能完成。此参数的值不能为空。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。

schemalocation
: VARCHAR (1000) 类型的输入参数，它可以具有空值，它指示要将 XML 模式文档添加至的 XML 模式主文档的模式位置。此参数是 XML 模式的外部名，即，可以在 XML 实例文档中使用 xsi:schemaLocation 属性来标识主文档。

content
: BLOB (30M) 类型的输入参数，它包含所添加的 XML 模式文档的内容。此参数的值不能为空；必须提供一个 XML 模式文档。

docproperty
: BLOB (5M) 类型的输入参数，它指示所添加的 XML 模式文档的属性。此参数可以具有空值；否则，值为 XML 文档。

示例:

```
CALL SYSPROC.XSR_ADDSCHEMADOC(
       'user1',
       'POschema',
       'http://myPOschema/address.xsd',
       :content_host_var,
       0)
```

XSR_COMPLETE

```
►►—XSR_COMPLETE—(—rschema—,—name—,—schemaproperties—,——————————►

►—isusedfordecomposition—)——————————————————————————————►◄
```

模式为 SYSPROC。

XSR_COMPLETE 过程是在 XML 模式注册过程中要调用的最后一个存储过程，它将向 XML 模式存储库（XSR）注册 XML 模式。在通过调用此存储过程来完成模式注册之前，将不能对 XML 模式进行验证。

权限

该过程的调用者的授权标识必须是 SYSCAT.XSROBJECTS 目录视图中记录的 XSR 对象的所有者。

rschema
: VARCHAR (128) 类型的输入参数，它指定 XML 模式的 SQL 模式。SQL 模式是用来在 XSR 中标识此 XML 模式的 SQL 标识的一部分，它将变成完整状态。（SQL 标识的另一部分是由 name 参数提供的。）此参数的值可以为空，这表示使用了缺省 SQL 模式（它是在专用寄存器 CURRENT SCHEMA 中定义的）。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。因为 XSR 对象与 XML 模式存储库外部的对象位于不同的名称空间中，所以 XSR 对象不会与存在于 XSR 外部的数据库对象之间发生名称冲突。

name
: VARCHAR(128) 类型的输入参数，它指定 XML 模式的名称。XML 模式的完整 SQL 标识（要对它执行完整性检查）为 *rschema.name*。必须通过调用 XSR_REGISTER 存储过程已经获得了 XML 模式名称，并且 XML 模式注册尚不能完成。此参数的值不能为空。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。

schemaproperties
: BLOB (5M) 类型的输入参数，它指定与 XML 模式相关联的属性（如果有）。如果没有相关联的属性，那么此参数的值为空；此参数的值也可以是一个用于表示 XML 模式属性的 XML 文档。

isusedfordecomposition
: Integer 类型的输入参数，它指示是否将 XML 模式用于分解。如果要将 XML 模式用于分解，那么应将此值设置为 1；否则应设置为 0。

示例:

```
CALL SYSPROC.XSR_COMPLETE(
       'user1',
       'POschema',
   :schemaproperty_host_var,
       0)
```

XSR_DTD

```
▶▶—XSR_DTD—(—rschema—,—name—,—systemid—,—publicid—,—content—)——————▶◀
```

模式为 SYSPROC。

XSR_DTD 过程用于向 XML 模式存储库（XSR）注册文档类型声明（DTD）。

权限

该过程的调用者的授权标识必须至少具有下列其中一种权限或特权:

- DBADM 权限。
- IMPLICIT_SCHEMA 数据库权限（如果 SQL 模式不存在）。
- CREATEIN 特权（如果 SQL 模式存在）。

rschema
: VARCHAR (128) 类型的输入和输出参数，它指定 DTD 的 SQL 模式。SQL 模式是用来在 XSR 中标识此 DTD 的 SQL 标识的一部分。（SQL 标识的另一部分是由 *name* 参数提供的。）此参数的值可以为空，这表示使用了缺省 SQL 模式（它是在专用寄存器 CURRENT SCHEMA 中定义的）。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。不能将以字符串"SYS"开头的关系模式用于此值。因为 XSR 对象与 XML 模式存储库外部的对象位于不同的名称空间中，所以 XSR 对象不会与存在于 XSR 外部的数据库对象之间发生名称冲突。

name
: VARCHAR (128) 类型的输入和输出参数，它指定 DTD 的名称。DTD 的完整 SQL 标识为 *rschema.name*，并且对于 XSR 中的所有对象应该是唯一的。此参数接受空值。当为此参数提供空值时，就会生成一个唯一值并存储在 XSR 中。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。

systemid
VARCHAR (1000) 类型的输入参数，它指定 DTD 的系统标识。DTD 的系统标识应该与 XML 实例文档的 DOCTYPE 声明或者 ENTITY 声明（如果使用的话，在前面添加关键字 SYSTEM 作为前缀）中的 DTD 的统一资源标识相匹配。此参数的值不能为空。系统标识可与公用标识同时指定。

publicid
VARCHAR (1000) 类型的输入参数，它指定 DTD 的公用标识。DTD 的公用标识应该与 XML 实例文档的 DOCTYPE 声明或者 ENTITY 声明（如果使用的话，在前面添加关键字 PUBLIC 作为前缀）中的 DTD 的统一资源标识相匹配。此参数接受空值，并且仅当在 XML 实例文档的 DOCTYPE 声明或者 ENTITY 声明中也指定了此参数时才使用它。

content
BLOB (30M) 类型的输入参数，它包含 DTD 文档的内容。此参数的值不能为空。

示例：注册由系统标识 *http://www.test.com/person.dtd* 和公用标识 *http://www.test.com/person* 标识的 DTD:

```
CALL SYSPROC.XSR_DTD ( 'MYDEPT' ,
      'PERSONDTD' ,
      'http://www.test.com/person.dtd' ,
      'http://www.test.com/person',
      :content_host_variable
   )
```

XSR_EXTENTITY

```
>>—XSR_EXTENTITY—(—rschema—,—name—,—systemid—,—publicid—,——————————>

>—content—)———————————————————————————————————————————————————————><
```

模式为 SYSPROC。

XSR_EXTENTITY 过程用于向 XML 模式存储库（XSR）注册外部实体。

权限

该过程的调用者的授权标识必须至少具有下列其中一种权限或特权：

- DBADM 权限。
- IMPLICIT_SCHEMA 数据库权限（如果 SQL 模式不存在）。
- CREATEIN 特权（如果 SQL 模式存在）。

rschema
VARCHAR (128) 类型的输入和输出参数，它指定外部实体的 SQL 模式。SQL 模式是用来在 XSR 中标识此外部实体的 SQL 标识的一部分。（SQL 标识的另一部分是由 *name* 参数提供的。）此参数的值可以为空，这表示使用了缺省 SQL 模式（它是在专用寄存器 CURRENT SCHEMA 中定义的）。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。不能将以字符串“SYS”开头的关系模式用于此值。因为 XSR 对象与 XML 模式存储库外部的对象位于不同的名称空间中，所以 XSR 对象不会与存在于 XSR 外部的数据库对象之间发生名称冲突。

name

VARCHAR(128) 类型的输入和输出参数，它指定外部实体的名称。外部实体的完整 SQL 标识为 *rschema.name*，并且对于 XSR 中的所有对象应该是唯一的。此参数接受空值。当为此参数提供空值时，就会生成一个唯一值并存储在 XSR 中。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。

systemid

VARCHAR (1000) 类型的输入参数，它指定外部实体的系统标识。外部实体的系统标识应该与 ENTITY 声明（如果使用的话，在前面添加关键字 SYSTEM 作为前缀）中的外部实体的统一资源标识相匹配。此参数的值不能为空。系统标识可与公用标识同时指定。

publicid

VARCHAR (1000) 类型的输入参数，它指定外部实体的公用标识。外部实体的公用标识应该与 ENTITY 声明（如果使用的话，在前面添加关键字 PUBLIC 作为前缀）中的外部实体的统一资源标识相匹配。此参数接受空值，并且仅当在 XML 实例文档的 DOCTYPE 声明或者 ENTITY 声明中也指定了此参数时才使用它。

content

BLOB (30M) 类型的输入参数，它包含外部实体文档的内容。此参数的值不能为空。

示例：注册由系统标识 *http://www.test.com/food/chocolate.txt* 和 *http://www.test.com/food/cookie.txt* 标识的外部实体:

```
CALL SYSPROC.XSR_EXTENTITY ( 'FOOD' ,
      'CHOCLATE' ,
      'http://www.test.com/food/chocolate.txt' ,
      NULL ,
      :content_of_chocolate.txt_as_a_host_variable
   )

CALL SYSPROC.XSR_EXTENTITY ( 'FOOD' ,
      'COOKIE' ,
      'http://www.test.com/food/cookie.txt' ,
      NULL ,
      :content_of_cookie.txt_as_a_host_variable
   )
```

XSR_UPDATE

```
►►—XSR_UPDATE—(—rschema1—,—name1—,—rschema2—,—name2—,——————————►

►–dropnewschema—)——————————————————————————————————————————————►◄
```

模式为 SYSPROC。

XSR_UPDATE 存储过程用于演进 XML 模式存储库（XSR）中的现有 XML 模式。这将允许您修改或扩展现有 XML 模式，以便可使用它来验证已存在或新插入的 XML 文档。

在调用 XSR_UPDATE 之前，必须在 XSR 中注册并完成指定为该过程参数的原始 XML 模式和新 XML 模式。这些 XML 模式还必须兼容。有关兼容性要求的详细信息，请参阅*演进 XML 模式的兼容性要求*。

权限

过程调用者的授权标识拥有的特权必须至少包括下列其中一项权限或特权:

- DBADM 权限。
- 对目录视图 SYSCAT.XSROBJECTS 和 SYSCAT.XSROBJECTCOMPONENTS 的 SELECT 特权以及下列其中一组特权:
  - XML 模式的所有者由 SQL 模式 *rschema1* 和对象名 *name1* 指定。
  - 对 *rschema1* 参数所指定的 SQL 模式的 ALTERIN 特权,以及在 *dropnewschema* 参数不等于零时,对 *rschema2* 参数所指定的 SQL 模式的 DROPIN 特权。

rschema1

VARCHAR (128) 类型的输入参数,它指定要更新的原始 XML 模式的 SQL 模式。SQL 模式是用来在 XSR 中标识此 XML 模式的 SQL 标识的一部分。(SQL 标识的另一部分由 *name1* 参数提供。)此参数的值不能为空。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。

name1

VARCHAR (128) 类型的输入参数,它指定要更新的原始 XML 模式的名称。XML 模式的完整 SQL 标识为 *rschema1.name1*。此 XML 模式必须已在 XSR 中注册并完成。此参数的值不能为空。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。

rschema2

VARCHAR (128) 类型的输入参数,它指定将用来更新原始 XML 模式的新 XML 模式的 SQL 模式。SQL 模式是用来在 XSR 中标识此 XML 模式的 SQL 标识的一部分。(SQL 标识的另一部分由 *name2* 参数提供。)此参数的值不能为空。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。

name2

VARCHAR (128) 类型的输入参数,它指定将用来更新原始 XML 模式的新 XML 模式的名称。XML 模式的完整 SQL 标识为 *rschema2.name2*。此 XML 模式必须已在 XSR 中注册并完成。此参数的值不能为空。应用于任何 SQL 标识的有效字符和定界符的规则也适用于此参数。

dropnewschema

Integer 类型的输入参数,它指示在使用新 XML 模式更新原始 XML 模式后是否应将新 XML 模式删除。将此参数设置为任何非零值将导致新的 XML 模式被删除。此参数的值不能为空。

示例:

```
CALL SYSPROC.XSR_UPDATE(
  'STORE',
  'PROD',
  'STORE',
  'NEWPROD',
  1)
```

XML 模式 STORE.PROD 的内容更新为 STORE.NEWPROD 的内容,并且 XML 模式 STORE.NEWPROD 被删除。

XSR 命令

REGISTER XMLSCHEMA

向 XML 模式存储库（XSR）注册 XML 模式。

权限

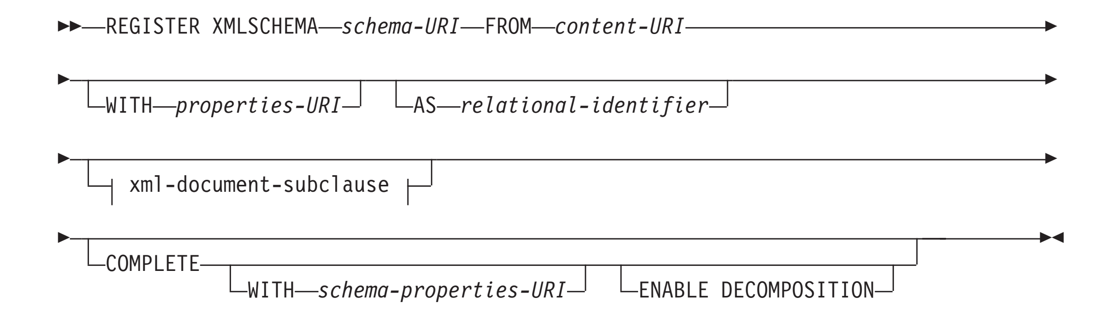
为下列其中一项:

- *dbadm*
- IMPLICIT_SCHEMA 数据库权限（如果 SQL 模式不存在）
- CREATEIN 特权（如果 SQL 模式存在）

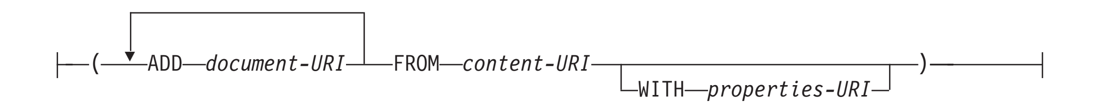
必需的连接

Database

命令语法



xml-document-subclause:



命令参数

schema-URI
: 指定正在注册的 XML 模式的 URI，供 XML 实例文档引用。

FROM *content-URI*
: 指定 XML 模式文档所在的 URI。仅支持由文件方案 URI 指定的本地文件。

WITH *properties-URI*
: 指定 XML 模式的属性文档的 URI。仅支持由文件方案 URI 指定的本地文件。

AS *relational-identifier*
: 指定可以用来表示正在注册的 XML 模式的名称。可以将关系名称指定为由两部分组成的 SQL 标识，这两部分分别是 SQL 模式和 XML 模式名，其格式

如下：SQLschema.name。如果未指定模式，那么使用缺省关系模式（它是在专用寄存器 CURRENT SCHEMA 中定义的）。如果未提供名称，那么将生成唯一值。

COMPLETE
指示不再添加 XML 模式文档。如果指定了此参数，那么会验证模式，并且在未发现任何错误时将该模式标记为可用。

WITH *schema-properties-URI*
指定 XML 模式的属性文档的 URI。仅支持由文件方案 URI 指定的本地文件。

ENABLE DECOMPOSITION
指定要将此模式用于分解 XML 文档。

ADD *document-URI*
指定要添加至此模式的 XML 模式文档的 URI，因为将从另一个 XML 文档中引用该文档。

FROM *content-URI*
指定 XML 模式文档所在的 URI。仅支持由文件方案 URI 指定的本地文件。

WITH *properties-URI*
指定 XML 模式的属性文档的 URI。仅支持由文件方案 URI 指定的本地文件。

示例

```
REGISTER XMLSCHEMA 'http://myPOschema/PO.xsd'
FROM 'file:///c:/TEMP/PO.xsd'
WITH 'file:///c:/TEMP/schemaProp.xml'
AS user1.POschema
```

使用说明

- 在可以引用 XML 模式文档并将它用于验证和注释之前，必须先向 XSR 注册它。此命令将执行 XML 模式注册过程的第一步，即，注册 XML 模式主文档。XML 模式注册过程的最后一步要求成功地对 XML 模式运行 COMPLETE XMLSCHEMA 命令。或者，如果不包括其他 XML 模式文档，那么在发出 REGISTER XMLSCHEMA 命令时附带 COMPLETE 关键字，以便通过一个步骤就完成注册过程。
- 当在数据库中注册 XML 模式时，根据该 XML 模式的大小，可能需要更大的应用程序堆（APPLHEAPSZ）。建议大小为 1024，但是更大的模式将需要更多内存。

ADD XMLSCHEMA DOCUMENT

在完成注册之前，将一个或多个 XML 模式文档添加至现有的不完整 XML 模式。

权限

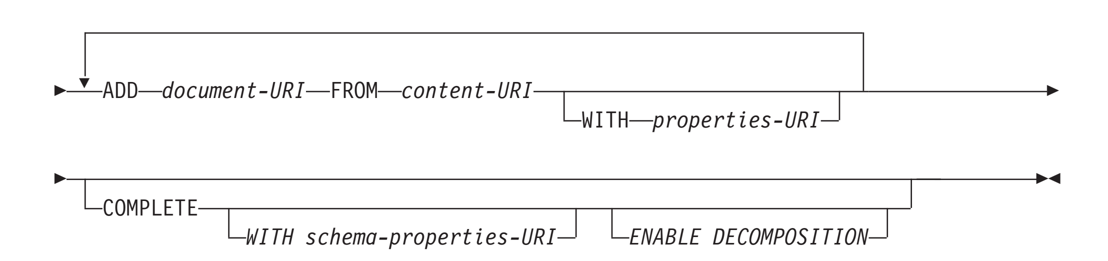
需要下列权限：

- 用户标识必须是 SYSCAT.XSROBJECTS 目录视图中记录的 XSR 对象的所有者。

必需的连接

Database

命令语法



描述

TO *relational-identifier*
指定要将其他模式文档添加至的已注册但是不完整的 XML 模式的关系名称。

ADD *document-URI*
指定要添加至此模式的 XML 模式文档的统一资源标识（URI），因为将从另一个 XML 文档中引用该文档。

FROM *content-URI*
指定 XML 模式文档所在的 URI。仅支持文件方案 URI。

WITH *properties-URI*
指定 XML 模式的属性文档的 URI。仅支持文件方案 URI。

COMPLETE
指示不再添加 XML 模式文档。如果指定了此参数，那么会验证模式，并且在未发现任何错误时将该模式标记为可用。

WITH *schema-properties-URI*
指定 XML 模式的属性文档的 URI。仅支持文件方案 URI。

ENABLE DECOMPOSITION
指定要将此模式用于分解 XML 文档。

示例

```
ADD XMLSCHEMA DOCUMENT TO JOHNDOE.PRODSCHEMA
      ADD 'http://myPOschema/address.xsd'
      FROM 'file:///c:/TEMP/address.xsd'
```

COMPLETE XMLSCHEMA

此命令将完成在 XML 模式存储库（XSR）中注册 XML 模式的过程。

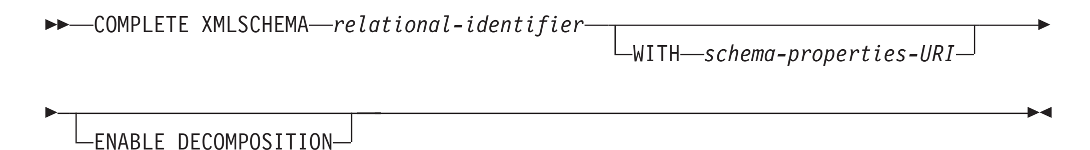
权限

- 用户标识必须是 SYSCAT.XSROBJECTS 目录视图中记录的 XSR 对象的所有者。

必需的连接

Database

命令语法



描述

relational-identifier
: 指定先前使用 REGISTER XMLSCHEMA 命令注册的 XML 模式的关系名称。可以将关系名称指定为由两部分组成的 SQL 标识，这两部分分别是 SQL 模式和 XML 模式名，其格式如下：*SQLschema.name*。如果未指定模式，那么使用缺省 SQL 模式（它是在专用寄存器 CURRENT SCHEMA 中定义的）。

WITH *schema-properties-URI*
: 指定 XML 模式的属性文档的统一资源标识（URI）。仅支持由文件方案 URI 指定的本地文件。只能在 XML 模式注册的完成阶段才能指定模式属性文档。

ENABLE DECOMPOSITION
: 指示可以将模式用于分解 XML 实例文档。

示例

```
COMPLETE XMLSCHEMA user1.POschema WITH 'file:///c:/TEMP/schemaProp.xml'
```

使用说明

在完成 XML 模式注册过程之前，不能引用 XML 模式或者将它用于验证或注释。此命令将完成由 REGISTER XMLSCHEMA 命令开始的 XML 模式的注册过程。

REGISTER XSROBJECT

在数据库目录中注册 XML 对象。受支持的对象是 DTD 和外部实体。

权限

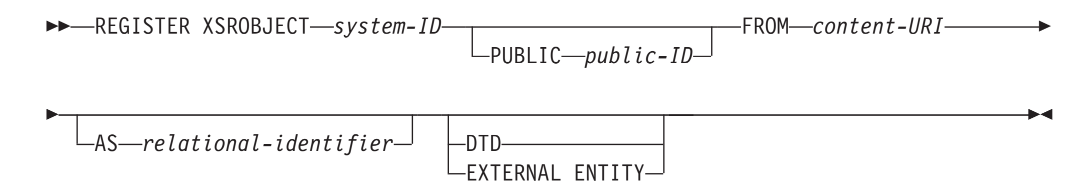
为下列其中一项:

- *dbadm*
- IMPLICIT_SCHEMA 数据库权限（如果 SQL 模式不存在）
- CREATEIN 特权（如果 SQL 模式存在）

必需的连接

Database

命令语法



命令参数

system-ID
指定在 XML 对象声明中指定的系统标识。

PUBLIC *public-ID*
在 XML 对象声明中指定一个可选的公用标识。

FROM *content-URI*
指定 XML 模式文档的内容所在的 URI。仅支持由文件方案 URI 指定的本地文件。

AS *relational-identifier*
指定可以用来表示正在注册的 XML 对象的名称。可以将关系名称指定为由两部分组成的 SQL 标识，这两部分分别是关系模式和名称，它们之间由句点分隔开。例如，“JOHNDOE.EMPLOYEEDTD”。如果未指定关系模式，那么会使用在专用寄存器 CURRENT SCHEMA 中定义的缺省关系模式。如果未指定名称，那么会自动生成一个名称。

DTD 指定正在注册的对象是“数据类型定义”（DTD）文档。

EXTERNAL ENTITY
指定正在注册的对象是外部实体。

示例

1. 提供以下样本 XML 文档，它引用了一个外部实体:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE copyright [
  <!ELEMENT copyright (#PCDATA)>

]>
<copyright>c</copyright>
```

需要注册该外部实体之后，才能成功地将此文档插入到 XML 列中。以下命令会注册一个实体，实体内容存储在 C:\TEMP 本地:

```
REGISTER XSROBJECT 'http://www.xmlwriter.net/copyright.xml'
      FROM 'c:\temp\copyright.xml' EXTERNAL ENTITY
```

2. 提供以下 XML 文档片段，它引用了一个 DTD:

```
<!--inform the XML processor
  that an external DTD is referenced-->
<?xml version="1.0" standalone="no" ?>

<!--define the location of the
  external DTD using a relative URL address-->
<!DOCTYPE document SYSTEM "http://www.xmlwriter.net/subjects.dtd">

<document>
```

```
  <title>Subjects available in Mechanical Engineering.</title>
    <subjectID>2.303</subjectID>
        <subjectname>Fluid Mechanics</subjectname>
 ...
```

需要注册该 DTD 之后，才能成功地将此文档插入到 XML 列中。以下命令会注册 DTD，DTD 定义存储在 C:\TEMP 本地并且与 DTD 相关联的关系标识为 “TEST.SUBJECTS”:

```
REGISTER XSROBJECT 'http://www.xmlwriter.net/subjects.dtd'
      FROM 'file:///c:/temp/subjects.dtd' AS TEST.SUBJECTS DTD
```

3. 提供以下样本 XML 文档，它引用了一个公用外部实体:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE copyright [
  <!ELEMENT copyright (#PCDATA)>

]>
<copyright>c</copyright>
```

需要注册该公用外部实体之后，才能成功地将此文档插入到 XML 列中。以下命令会注册一个实体，实体内容存储在 C:\TEMP 本地:

```
REGISTER XSROBJECT 'http://www.w3.org/xmlspec/copyright.xml'
      PUBLIC '-//W3C//TEXT copyright//EN' FROM 'file:///c:/temp/copyright.xml'
      EXTERNAL ENTITY
```

UPDATE XMLSCHEMA

在 XML 模式存储库（XSR）中使用一种 XML 模式更新另一种 XML 模式。

权限

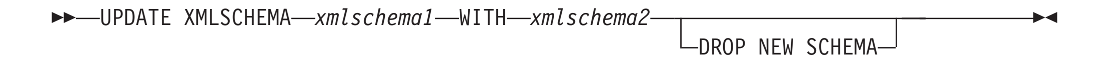
为下列其中一项:

- *dbadm*
- 对目录视图 SYSCAT.XSROBJECTS 和 SYSCAT.XSROBJECTCOMPONENTS 的 SELECT 特权以及下列其中一组特权:
  - 对要更新的 XML 模式的 ALTERIN 特权以及对新 XML 模式的 DROPIN 特权（如果指定了 DROP NEW SCHEMA 选项）。
  - xmlschema1 指定的 XML 模式的 OWNER。

必需的连接

Database

命令语法



命令参数

UPDATE XMLSCHEMA *xmlschema1*
: 指定要更新的原始 XML 模式的 SQL 标识。

WITH *xmlschema2*
指定将用来更新原始 XML 模式的新 XML 模式的 SQL 标识。

DROP NEW SCHEMA
指示在使用新 XML 模式更新原始 XML 模式后应将新 XML 模式删除。

示例

```
UPDATE XMLSCHEMA JOHNDOE.OLDPROD
WITH JOHNDOE.NEWPROD
DROP NEW SCHEMA
```

XML 模式 JOHNDOE.OLDPROD 的内容更新为 JOHNDOE.NEWPROD 的内容，并且 XML 模式 JOHNDOE.NEWPROD 被删除。

使用说明

- 原始 XML 模式和新 XML 模式必须兼容。有关兼容性要求的详细信息，请参阅『演进 XML 模式的兼容性要求』。
- 在更新 XML 模式之前，必须在 XML 模式存储库（XSR）中注册原始模式和新模式。

附录 D. DB2 技术信息概述

可以通过下列工具和方法获取 DB2 技术信息:

- DB2 信息中心
  - 主题(任务、概念和参考主题)
  - DB2 工具的帮助
  - 样本程序
  - 教程
- DB2 书籍
  - PDF 文件(可下载)
  - PDF 文件(在 DB2 PDF DVD 中)
  - 印刷版书籍
- 命令行帮助
  - 命令帮助
  - 消息帮助

注: DB2 信息中心主题的更新频率比 PDF 书籍或硬拷贝书籍的更新频率高。要获取最新信息,请安装可用的文档更新,或者参阅 ibm.com 上的 DB2 信息中心。

可以在线访问 ibm.com 上的其他 DB2 技术信息,如技术说明、白皮书和 IBM Redbooks® 出版物。访问位于以下网址的 DB2 信息管理软件库站点: http://www.ibm.com/software/data/sw-library/。

文档反馈

我们非常重视您对 DB2 文档的反馈。如果您想就如何改善 DB2 文档提出建议,请将电子邮件发送至 db2docs@ca.ibm.com。 DB2 文档小组会阅读您的所有反馈,但不能直接答复您。请尽可能提供具体的示例,这样我们才能更好地了解您所关心的问题。如果您要提供有关具体主题或帮助文件的反馈,请加上标题和 URL。

请不要用以上电子邮件地址与 DB2 客户支持机构联系。如果您遇到文档不能解决的 DB2 技术问题,请与您当地的 IBM 服务中心联系以获得帮助。

硬拷贝或 PDF 格式的 DB2 技术库

下列各表描述 IBM 出版物中心(网址为 www.ibm.com/shop/publications/order)提供的 DB2 资料库。可从 www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 下载 PDF 格式的 DB2 版本 9.7 手册的英文版本和翻译版本。

尽管这些表标识书籍有印刷版,但可能未在您所在国家或地区提供。

每次更新手册时,表单号都会递增。确保您正在阅读下面列示的手册的最新版本。

注: *DB2* 信息中心的更新频率比 PDF 或硬拷贝书籍的更新频率高。

表 86. DB2 技术信息

书名	书号	是否提供印刷版	最近一次更新时间
Administrative API Reference	SC27-2435-00	是	2009 年 8 月
Administrative Routines and Views	SC27-2436-00	否	2009 年 8 月
Call Level Interface Guide and Reference, Volume 1	SC27-2437-00	是	2009 年 8 月
Call Level Interface Guide and Reference, Volume 2	SC27-2438-00	是	2009 年 8 月
Command Reference	SC27-2439-00	是	2009 年 8 月
《数据移动指南和参考》	S151-1186-00	是	2009 年 8 月
《数据恢复及高可用性指南与参考》	S151-1187-00	是	2009 年 8 月
《数据库管理概念和配置参考》	S151-1163-00	是	2009 年 8 月
《数据库监视指南和参考》	S151-1165-00	是	2009 年 8 月
《数据库安全性指南》	S151-1188-00	是	2009 年 8 月
DB2 Text Search Guide	SC27-2459-00	是	2009 年 8 月
《开发 ADO.NET 和 OLE DB 应用程序》	S151-1167-00	是	2009 年 8 月
《开发嵌入式 SQL 应用程序》	S151-1168-00	是	2009 年 8 月
Developing Java Applications	SC27-2446-00	是	2009 年 8 月
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-2447-00	否	2009 年 8 月
Developing User-defined Routines (SQL and External)	S151-1169-00	是	2009 年 8 月
Getting Started with Database Application Development	G151-1170-00	是	2009 年 8 月
《Linux 和 Windows 上的 DB2 安装和管理入门》	G151-1172-00	是	2009 年 8 月
《全球化指南》	S151-1189-00	是	2009 年 8 月
《安装 DB2 服务器》	G151-1174-00	是	2009 年 8 月
《安装 IBM 数据服务器客户端》	G151-1175-00	否	2009 年 8 月
《消息参考第 1 卷》	S151-1182-00	否	2009 年 8 月

表 86. DB2 技术信息 (续)

书名	书号	是否提供印刷版	最近一次更新时间
《消息参考第 2 卷》	S151-1183-00	否	2009 年 8 月
《Net Search Extender 管理和用户指南》	S151-1185-00	否	2009 年 8 月
《分区和集群指南》	S151-1190-00	是	2009 年 8 月
《pureXML 指南》	S151-1180-00	是	2009 年 8 月
Query Patroller Administration and User's Guide	SC27-2467-00	否	2009 年 8 月
《Spatial Extender 和地理数据管理功能部件用户指南和参考》	SC27-2468-00	否	2009 年 8 月
《SQL 过程语言：应用程序启用和支持》	S151-1171-00	是	2009 年 8 月
SQL Reference, Volume 1	SC27-2456-00	是	2009 年 8 月
SQL Reference, Volume 2	SC27-2457-00	是	2009 年 8 月
《故障诊断和调整数据库性能》	S151-1164-00	是	2009 年 8 月
《升级到 DB2 版本 9.7》	S151-1173-00	是	2009 年 8 月
《Visual Explain 教程》	S151-1184-00	否	2009 年 8 月
《DB2 版本 9.7 新增内容》	S151-1179-00	是	2009 年 8 月
Workload Manager Guide and Reference	SC27-2464-00	是	2009 年 8 月
《XQuery 参考》	S151-1181-00	否	2009 年 8 月

表 87. 特定于 DB2 Connect 的技术信息

书名	书号	是否提供印刷版	最近一次更新时间
《安装和配置 DB2 Connect 个人版》	S151-1177-00	是	2009 年 8 月
《安装和配置 DB2 Connect 服务器》	S151-1178-00	是	2009 年 8 月
《DB2 Connect 用户指南》	S151-1176-00	是	2009 年 8 月

表 88. Information Integration 技术信息

书名	书号	是否提供印刷版	最近一次更新时间
Information Integration: Administration Guide for Federated Systems	SC19-1020-02	是	2009 年 8 月
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-04	是	2009 年 8 月

表 88. *Information Integration* 技术信息 (续)

书名	书号	是否提供印刷版	最近一次更新时间
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-02	否	2009 年 8 月
Information Integration: SQL Replication Guide and Reference	SC19-1030-02	是	2009 年 8 月
Information Integration: Introduction to Replication and Event Publishing	GC19-1028-02	是	2009 年 8 月

订购印刷版的 DB2 书籍

如果您需要印刷版的 DB2 书籍，可以在许多（但不是所有）国家或地区在线购买。无论何时都可以从当地的 IBM 代表处订购印刷版的 DB2 书籍。请注意，*DB2 PDF 文档* DVD 上的某些软拷贝书籍没有印刷版。例如，*DB2 消息参考*的任何一卷都没有提供印刷版书籍。

只要支付一定费用，就可以从 IBM 获取 DB2 PDF 文档 DVD，该 DVD 包含许多 DB2 书籍的印刷版。 根据您下订单的位置，您可能能够从 IBM 出版物中心在线订购书籍。如果在线订购在您所在国家或地区不可用，您始终可以从当地的 IBM 代表处订购印刷版 DB2 书籍。注意，并非 DB2 PDF 文档 DVD 上的所有书籍都有印刷版。

注：最新最完整的 DB2 文档保留在 DB2 信息中心中，网址为：http://publib.boulder.ibm.com/infocenter/db2luw/v9r7。

要订购印刷版的 DB2 书籍：

- 要了解您是否可从所在国家或地区在线订购印刷版的 DB2 书籍，可查看 IBM 出版物中心站点，网址为：http://www.ibm.com/shop/publications/order。必须先选择国家、地区或语言才能访问出版物订购信息，然后再按照针对您所在位置的订购指示信息进行订购。
- 要从当地的 IBM 代表处订购印刷版的 DB2 书籍：
  1. 从下列其中一个 Web 站点找到当地代表处的联系信息：
     - IBM 全球联系人目录，网址为 www.ibm.com/planetwide。
     - IBM 出版物 Web 站点，网址为 http://www.ibm.com/shop/publications/order。必须先选择国家、地区或语言才能访问对应您的所在地的出版物主页。在此页面中访问“关于此站点”链接。
  2. 请在致电时说明您想订购 DB2 出版物。
  3. 请向您当地的代表提供想要订购的书籍的书名和书号。有关书名和书号的信息，请参阅第 437 页的『硬拷贝或 PDF 格式的 DB2 技术库』。

从命令行处理器显示 SQL 状态帮助

DB2 产品针对可能充当 SQL 语句结果的条件返回 SQLSTATE 值。SQLSTATE 帮助说明 SQL 状态和 SQL 状态类代码的含义。

要启动 SQL 状态帮助，请打开命令行处理器并输入：

```
? sqlstate or ? class code
```

其中，*sqlstate* 表示有效的 5 位 SQL 状态，*class code* 表示该 SQL 状态的前 2 位。例如，`? 08003` 显示 08003 SQL 状态的帮助，而 `? 08` 显示 08 类代码的帮助。

访问不同版本的 DB2 信息中心

对于 DB2 版本 9.7 主题，DB2 信息中心 URL 为 http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/

对于 DB2 版本 9.5 主题，DB2 信息中心 URL 为 http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/

对于 DB2 版本 9 主题，DB2 信息中心 URL 为 http://publib.boulder.ibm.com/infocenter/db2luw/v9/

对于 DB2 版本 8 主题，请访问版本 8 信息中心 URL：http://publib.boulder.ibm.com/infocenter/db2luw/v8/

在 DB2 信息中心中以您的首选语言显示主题

DB2 信息中心尝试以您在浏览器首选项中指定的语言显示主题。如果未提供主题的首选语言翻译版本，那么 DB2 信息中心将显示该主题的英文版。

- 要在 Internet Explorer 浏览器中以您的首选语言显示主题：
  1. 在 Internet Explorer 中，单击**工具** —> **Internet 选项** —> **语言...** 按钮。"语言首选项"窗口打开。
  2. 确保您的首选语言被指定为语言列表中的第一个条目。
     - 要将新语言添加至列表，请单击**添加...** 按钮。

       注：添加语言并不能保证计算机具有以首选语言显示主题所需的字体。
     - 要将语言移至列表顶部，请选择该语言并单击**上移**按钮直到该语言成为语言列表中的第一个条目。
  3. 清除浏览器高速缓存然后刷新页面以便以首选语言显示 DB2 信息中心。
- 要在 Firefox 或 Mozilla 浏览器中以首选语言显示主题：
  1. 在**工具** —> **选项** —> **高级**对话框中的**语言**部分中选择按钮。"语言"面板将显示在"首选项"窗口中。
  2. 确保您的首选语言被指定为语言列表中的第一个条目。
     - 要将新语言添加至列表，请单击**添加...** 按钮以从"添加语言"窗口中选择一种语言。
     - 要将语言移至列表顶部，请选择该语言并单击**上移**按钮直到该语言成为语言列表中的第一个条目。

3. 清除浏览器高速缓存然后刷新页面以便以首选语言显示 DB2 信息中心。

在某些浏览器和操作系统组合上，可能还必须将操作系统的区域设置更改为您选择的语言环境和语言。

更新安装在您的计算机或内部网服务器上的 DB2 信息中心

本地安装的 DB2 信息中心必须定期进行更新。

开始前

必须已安装 DB2 版本 9.7 信息中心。有关详细信息，请参阅《安装 DB2 服务器》中的“使用 DB2 安装向导来安装 DB2 信息中心”主题。所有适用于安装信息中心的先决条件和限制同样适用于更新信息中心。

关于此任务

可自动或手动更新现有 DB2 信息中心:

- 自动更新 – 更新现有信息中心功能和语言。自动更新的一个优点是在更新期间，信息中心不可用的时间最短。另外，自动更新可设置为作为定期运行的其他批处理作业的一部分运行。
- 手动更新 – 应该在更新过程期间要添加功能或语言时使用。例如，如果本地信息中心最初安装的是英语和法语版，而现在还要安装德语版；那么手动更新将安装德语版，并更新现有信息中心的功能和语言。但是，手动更新要求您手动停止、更新和重新启动信息中心。在整个更新过程期间信息中心不可用。

过程

此主题详细说明了自动更新的过程。有关手动更新的指示信息，请参阅“手动更新安装在您的计算机或内部网服务器上的 DB2 信息中心”主题。

要自动更新安装在您的计算机或内部网服务器上的 DB2 信息中心:

1. 在 Linux 操作系统上,
   a. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 /opt/ibm/db2ic/V9.7 目录中。
   b. 从安装目录浏览至 doc/bin 目录。
   c. 运行 ic-update 脚本:
      ```
      ic-update
      ```
2. 在 Windows 操作系统上,
   a. 打开命令窗口。
   b. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 <Program Files>\IBM\DB2 Information Center\Version 9.7 目录中，其中 <Program Files> 表示 Program Files 目录的位置。
   c. 从安装目录浏览至 doc\bin 目录。
   d. 运行 ic-update.bat 文件:
      ```
      ic-update.bat
      ```

结果

DB2 信息中心自动重新启动。如果更新可用，那么信息中心会显示新的以及更新后的主题。如果信息中心更新不可用，那么会在日志中添加消息。日志文件位于 doc\eclipse\configuration 目录中。日志文件名称是随机生成的编号。例如，1239053440785.log。

手动更新安装在您的计算机或内部网服务器上的 DB2 信息中心

如果已经在本地安装了 DB2 信息中心，那么您可以从 IBM 获取文档更新并安装。

手动更新在本地安装的 DB2 信息中心要求您:

1. 停止计算机上的 DB2 信息中心，然后以独立方式重新启动信息中心。如果以独立方式运行信息中心，那么网络上的其他用户将无法访问信息中心，因而您可以应用更新。DB2 信息中心的工作站版本总是以独立方式运行。
2. 使用"更新"功能部件来查看可用的更新。如果有您必须安装的更新，那么请使用"更新"功能部件来获取并安装这些更新。

   注: 如果您的环境要求在一台未连接至因特网的机器上安装 DB2 信息中心更新，那么通过使用一台已连接至因特网并有已安装的 DB2 信息中心的机器将更新站点镜像至本地文件系统。如果网络中有许多用户将安装文档更新，那么可以通过在本地也为更新站点制作镜像并为更新站点创建代理来缩短每个人执行更新所需要的时间。
   如果提供了更新包，请使用"更新"功能部件来获取这些更新包。但是，只有在单机方式下才能使用"更新"功能部件。
3. 停止独立信息中心，然后在计算机上重新启动 DB2 信息中心。

注: 在 Windows 2008、Windows Vista 和更高版本上，稍后列示在此部分的命令必须作为管理员运行。要打开具有全面管理员特权的命令提示符或图形工具，请右键单击快捷方式，然后选择**以管理员身份运行**。

要更新安装在您的计算机或内部网服务器上的 DB2 信息中心:

1. 停止 DB2 信息中心。
   - 在 Windows 上，单击**开始** → **控制面板** → **管理工具** → **服务**。右键单击 **DB2 信息中心**服务，并选择**停止**。
   - 在 Linux 上，输入以下命令:
     ```
     /etc/init.d/db2icdv97 stop
     ```
2. 以独立方式启动信息中心。
   - 在 Windows 上:
     a. 打开命令窗口。
     b. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 <Program Files>\IBM\DB2 Information Center\Version 9.7 目录中，其中 `<Program Files>` 表示 Program Files 目录的位置。
     c. 从安装目录浏览至 doc\bin 目录。
     d. 运行 help_start.bat 文件:
     ```
     help_start.bat
     ```
   - 在 Linux 上:
     a. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 /opt/ibm/db2ic/V9.7 目录中。
     b. 从安装目录浏览至 doc/bin 目录。

c. 运行 help_start 脚本:

```
help_start
```

系统缺省 Web 浏览器将打开以显示独立信息中心。

3. 单击**更新**按钮（ ）。（必须在浏览器中启用 JavaScript™。） 在信息中心的右边面板上，单击**查找更新**。 将显示现有文档的更新列表。
4. 要启动安装进程，请检查您要安装的选项，然后单击**安装更新**。
5. 在安装进程完成后，请单击**完成**。
6. 要停止独立信息中心，请执行下列操作:
   - 在 Windows 上，浏览至安装目录的 doc\bin 目录并运行 help_end.bat 文件:

     ```
     help_end.bat
     ```

     注: help_end 批处理文件包含安全地停止使用 help_start 批处理文件启动的进程所需的命令。不要使用 Ctrl-C 或任何其他方法来停止 help_start.bat。
   - 在 Linux 上，浏览至安装目录的 doc/bin 目录并运行 help_end 脚本:

     ```
     help_end
     ```

     注: help_end 脚本包含安全地停止使用 help_start 脚本启动的进程所需的命令。不要使用任何其他方法来停止 help_start 脚本。
7. 重新启动 DB2 信息中心。
   - 在 Windows 上，单击**开始** → **控制面板** → **管理工具** → **服务**。右键单击 **DB2 信息中心**服务，并选择**启动**。
   - 在 Linux 上，输入以下命令:

     ```
     /etc/init.d/db2icdv97 start
     ```

更新后的 DB2 信息中心将显示新的以及更新后的主题。

DB2 教程

DB2 教程帮助您了解 DB2 产品的各个方面。这些课程提供了逐步指示信息。

开始之前

可从信息中心查看 XHTML 版的教程: http://publib.boulder.ibm.com/infocenter/db2help/ 。

某些课程使用了样本数据或代码。有关其特定任务的任何先决条件的描述，请参阅教程。

DB2 教程

要查看教程，请单击标题。

《pureXML 指南》中的“pureXML”
: 设置 DB2 数据库以存储 XML 数据以及对本机 XML 数据存储执行基本操作。

《Visual Explain 教程》中的“Visual Explain”
: 使用 Visual Explain 来分析、优化和调整 SQL 语句以获取更好的性能。

DB2 故障诊断信息

提供了很多故障诊断和问题确定信息以帮助您使用 DB2 数据库产品。

DB2 文档

故障诊断信息可在《DB2 故障诊断指南》或 *DB2 信息中心*的“数据库基础”部分中找到。可在该处找到有关如何使用 DB2 诊断工具和实用程序来隔离和找出问题的信息、某些最常见问题的解决方案以及有关如何解决使用 DB2 数据库产品时可能遇到的问题的建议。

DB2 技术支持 Web 站点

如果您遇到了问题并且想要获取查找可能的原因和解决方案的帮助，请参阅 DB2 技术支持 Web 站点。该“技术支持”站点具有指向最新 DB2 出版物、技术说明、授权程序分析报告（APAR 或错误修订）、修订包和其他资源的链接。可搜索此知识库并查找问题的可能解决方案。

请访问 DB2 技术支持 Web 站点：http://www.ibm.com/software/data/db2/support/db2_9/。

条款和条件

如果符合以下条款和条件，那么授予您使用这些出版物的许可权。

个人使用： 只要保留所有的专有权声明，您就可以为个人、非商业使用复制这些出版物。未经 IBM 明确同意，您不可以分发、展示或制作这些出版物或其中任何部分的演绎作品。

商业使用： 只要保留所有的专有权声明，您就可以仅在企业内复制、分发和展示这些出版物。未经 IBM 明确同意，您不可以制作这些出版物的演绎作品，或者在您的企业外部复制、分发或展示这些出版物或其中的任何部分。

除非本许可权中明确授予，否则不得授予对这些出版物或其中包含的任何信息、数据、软件或其他知识产权的任何许可权、许可证或权利，无论是明示的还是暗含的。

当使用这些出版物损害了 IBM 的利益，或者根据 IBM 的规定，未正确遵守上述指导说明时，那么 IBM 保留自主决定撤销本文授予的许可权的权利。

只有您完全遵循所有适用的法律和法规，包括所有的美国出口法律和法规，您才可以下载、出口或再出口该信息。

IBM 对这些出版物的内容不作任何保证。这些出版物“按现状”提供，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的关于适销和适用于某种特定用途的保证。

附录 E. 声明

本信息是为在美国提供的产品和服务编写的。有关非 IBM 产品的信息是基于首次出版此文档时的可获信息且会随时更新。

IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节字符集（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区: International Business Machines Corporation"按现状"提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是此 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的：（i）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

只要遵守适当的条款和条件，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息可能包含在日常业务操作中使用的数据和报告的示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名称都是虚构的，与实际商业企业所用的名称和地址的任何雷同纯属巧合。

版权许可:

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。此样本程序“按现状”提供，且不附有任何种类的保证。对于使用此样本程序所引起的任何损坏，IBM 将不承担责任。

凡这些样本程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明:

©（*贵公司的名称*）（*年份*）. 此部分代码是根据 IBM 公司的样本程序衍生出来的。© Copyright IBM Corp. （*输入年份*）. All rights reserved.

商标

IBM、IBM 徽标和 ibm.com® 是 International Business Machines Corp. 在全球范围许多管辖区域内的商标或注册商标。其他产品和服务名称可能是 IBM 或其他公司的商标。Web 站点 www.ibm.com/legal/copytrade.shtml 上的“版权和商标信息”中提供了 IBM 商标的最新列表。

下列术语是其他公司的商标或注册商标

- Linux 是 Linus Torvalds 在美国和/或其他国家或地区的注册商标。
- Java 和所有基于 Java 的商标和徽标是 Sun Microsystems,Inc. 在美国和/或其他国家或地区的商标。
- UNIX 是 The Open Group 在美国和其他国家或地区的注册商标。
- Intel®、Intel 徽标、Intel Inside®、Intel Inside 徽标、Intel® Centrino®、Intel Centrino 徽标、Celeron®、Intel® Xeon®、Intel SpeedStep®、Itanium® 和 Pentium® 是 Intel 公司或其子公司在美国和其他国家或地区的商标或注册商标。
- Microsoft、Windows、Windows NT® 和 Windows 徽标是 Microsoft Corporation 在美国和/或其他国家或地区的商标。

其他公司、产品或服务名称可能是其他公司的商标或服务标记。

索引

[B]

[C]

[D]

[F]

[G]

[H]

[J]

[K]

[L]

[M]

[N]

[Q]

[S]

[T]

[W]

[X]

[Y]

[Z]

[数字]

U

V

X

[特别字符]

IBM

Printed in China

S151-1180-00



Spine information:

IBM DB2 9.7 Linux 版、UNIX 版和 Windows 版 **pureXML 指南** IBM